Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Nejc Kokalj

Zajem in vizualizacija podatkov z mobilno aplikacijo v brezžičnem senzorskem omrežju

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: viš. pred. dr. Robert Rozman
Somentor: doc. dr. Anton Biasizzo

Ljubljana, 2018

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Vedno več časa preživimo v najrazličnejših zgradbah. Prav tako smo vse bolj trajnostno in ekološko osveščeni in poskušamo v teh zaprtih življenjskih okoljih vzdrževati človeku prijazne razmere na čimbolj cenovno in trajnostno vzdržen način. Prvi običajni korak v tej smeri predstavlja ustrezen sistem za zaznavo in nadzor okoljskih parametrov v zgradbah. Izdelajte delujoč prototip takega sistema. Kot osnovo uporabite obstoječi sistem povezanih brezžičnih senzorjev in ga ustrezno nadgradite. Napišite tudi potrebno programsko opremo za podporni strežnik in mobilno aplikacijo, ki bo zajete in shranjene podatke ustrezno predstavila uporabnikom. Ob tem zagotovite karseda prijetno uporabniško izkušnjo in primerno informativnost sistema.

Zahvaljujem se družini za omogočanje študija, mentorjema za pomoč, napotke in predloge ob pisanju diplomskega dela in Institutu „Jožef Stefan“ za možnost opravljanja diplome v somentorstvu na konkretnem primeru.

Kazalo

Povzetek

Abstract

1 Uvod **1**

1.1 Predstavitev BLE multi-hop mehanizma 3

2 Uporabljena strojna oprema **5**

2.1 Brezžični multisenzor IJS 5

2.2 Mikroračunalnik Raspberry Pi 8

3 Nadgradnja brezžičnih multisenzorjev IJS **11**

3.1 Priključitev senzorja temperature in vlage 11

3.2 Vključitev novega senzorja v programsko opremo 12

3.3 Odprava pomanjkljivosti v programski opremi 17

4 Postavitev spletnega strežnika **21**

4.1 Podatkovna zbirka 21

4.2 Skripta za zajemanje podatkov 23

4.3 Spletna storitev 24

5 Razvoj mobilne aplikacije **29**

5.1 Prijava 31

5.2 Pridobivanje podatkov o vremenu 32

5.3 Pregled sob 34

5.4 Pregled zgodovine 42
5.5 Statusni pogled 45
5.6 Nastavitve 48

6 Sklepne ugotovitve **53**

Literatura **55**

Seznam uporabljenih kratic

kratica	angleško	slovensko
LAMP	Linux Apache MySQL PHP	Linux Apache MySQL PHP
SQL	structured query language	strukturirani povpraševalni jezik
UART	universal asynchronous receiver-transmitter	vezje za serijski prenos podatkov
SOIC	small outline integrated circuit	majhno podnožje integriranega vezja
I²C	inter-integrated circuit	serijsko vodilo integriranih vezij
SPI	serial peripheral interface	serijski periferni vmesnik
JSON	javascript object notation	notacija objekta javascript
MAC	media access control	nadzor dostopa do povezave
BLE	bluetooth low energy	nizko energijski bluetooth
API	application programming interface	aplikacijski programski vmesnik
YQL	yahoo query language	yahoo povpraševalni jezik
PHP	PHP: Hypertext Preprocessor	predprocesor nadbesedila PHP
HTML	hyper text markup language	jezik za označevanje nadbesedila
REST	representational state transfer	prenos s predstavljenim stanjem
PNG	portable network graphics	prenosljiva spletna grafika
URI	uniform resource identifier	enotni identifikator vira
HSV	hue saturation value	odtenek nasičenost svetlost
RGB	red green blue	rdeča zelena modra

Povzetek

Naslov: Zajem in vizualizacija podatkov z mobilno aplikacijo v brezžičnem senzorskem omrežju

Avtor: Nejc Kokalj

V okviru naloge smo izdelali mobilno aplikacijo za vizualizacijo stanja brezžičnih senzorjev. Uporabili smo brezžično senzorsko omrežje, razvito na Institutu „Jožef Stefan“. Brezžične senzorje smo nadgradili z možnostjo merjenja vlage. Ustrezno smo nadgradili tudi mehanizem za delovanje senzorskega omrežja. Med nadgradnjo smo opazili pomanjkljivost v delovanju senzorskega omrežja in jo odpravili. Za podporo mobilne aplikacije smo postavili zaledni strežnik na mikroračunalniku Raspberry Pi. Zaledni strežnik vključuje skripto, ki komunicira s senzorskim omrežjem, podatkovno zbirko in spletni programski vmesnik API. Mobilno aplikacijo smo razvili v okolju Android Studio. Aplikacija omogoča prikaz poljubnega tlorisa stavbe obogatenega z barvami, ki predstavljajo razporeditev temperature in vlage v prostorih glede na lokacije sten in senzorjev. V aplikacijo smo tudi vgradili možnost ogleda zgodovinskih podatkov s prikazom v grafu. Strežnik smo povezali v internet in mobilno aplikacijo preizkusili na mobilni napravi.

Ključne besede: BLE, multi-hop, BGScript, Raspberry Pi, Android.

Abstract

Title: Wireless Sensor Network Data Acquisition and Visualization using Mobile Application

Author: Nejc Kokalj

This thesis presents the process of creating a mobile application for visualizing wireless sensor data. We have used the existing wireless sensor network, developed at "Jožef Stefan" institute. The sensor network nodes were upgraded with the ability to measure relative humidity. The sensor network mechanism was upgraded to allow humidity readings as well. During the upgrade, we noticed a deficiency in the sensor network mechanism and improved it. In order to support the mobile application, we developed a back-end server based on a Raspberry Pi microcomputer. The back-end server includes a script which communicates with the sensor network, a database and a web API. The mobile application was developed using Android studio. The application displays the sensors on a floor plan with colors representing the temperature or humidity. The floor plan and position of the sensors may be customized. The application also displays graphs for viewing historical data. The platform was connected to the internet and tested on a mobile device.

Keywords: BLE, multi-hop, BGScript, Raspberry Pi, Android.

Poglavje 1

Uvod

Tehnologija BLE (angl. Bluetooth Low Energy) je nizkoenergetska tehnologija brezžičnega prenosa podatkov. Namenjena je predvsem nadzoru naprav kratkega dosega. Pričakuje se, da bo BLE v prihodnjih nekaj letih vgrajen v miljarde naprav [5]. Prav tako se vse več uporabljajo brezžična senzorska omrežja. Ta za komunikacijo uporabljajo različne brezžične tehnologije in topologije. Senzorji takšnih omrežij so čim manjši, poceni in imajo nizko porabo električne enrgije, saj se pogosto uporabljajo z baterijskim napajanjem.

Za razliko od primerljivih brezžičnih tehnologij kot je Zigbee, BLE podpira samo topologiji zvezde in P2P. Na Institutu „Jožef Stefan“ so v okviru članka "Multi-hop komunikacija v Bluetooth Low Energy ad-hoc brezžičnem senzorskem omrežju" [9] razvili BLE 4.0 mehanizem za multi-hop[1] prenos podatkov v centralno vozlišče. Prednost takšne topologije je, da lahko pridobimo podatke tudi iz vozlišč, ki niso v neposrednem dosegu. Mehanizem je bolj podrobno opisan v poglavju 1.1. V okviru diplomskega dela smo mehanizem in pripadajočo strojno opremo nadgradili z možnostjo merjenja vlage. Razvili smo spletno platformo, ki podatke zajame iz brezžičnega senzorskega omrežja in jih shrani v podatkovno zbirko. Spletna platforma podatke posreduje preko vmesnika API, ki podatke vrača v JSON obliki. Razvili smo tudi mobilno aplikacijo, ki omogoča vizualizacijo podatkov na različne načine.

[1] Za izraz multi-hop nismo našli uporabnega prevoda, zato bo uporabljen prvoten izraz.

Diplomsko delo je sestavljeno iz naslednjih poglavij. V 2. poglavju je opisana uporabljena strojna oprema. Poglavje je razdeljeno na dve podpoglavji. V prvem podpoglavju smo opisali senzorsko strojno opremo, v drugem pa strežnik Raspberry Pi.

V 3. poglavju je opisan postopek nadgradnje brezžičnih senzorjev z novo strojno in programsko opremo. Razvojne plošče brezžičnih senzorjev smo nadgradili z možnostjo branja vlage, za kar smo uporabili Honeywellov digitalni senzor serije HIH6100. Predstavljen je tudi postopek nadgradnje vgrajene programske opreme brezžičnih senzorjev za podporo novega senzorja. V tem poglavju so opisane tudi spremembe v multi-hop mehanizmu. Med nadgradnjo programske opreme smo naleteli na nekaj pomanjkljivosti v vgrajeni programski opremi, kar smo odpravili.

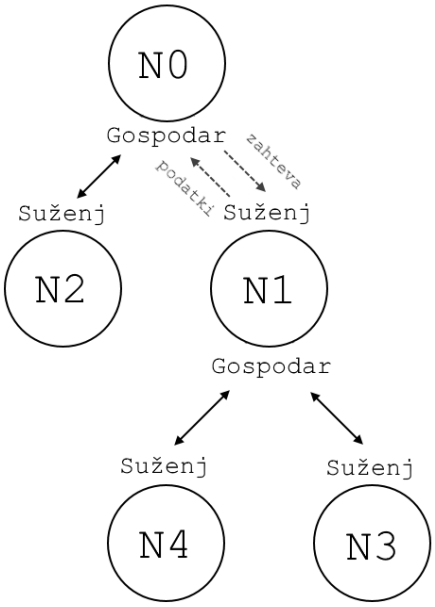
V 4. poglavju je opisana postavitev spletnega strežnika na mikroračunalniku Raspberry Pi. Na mikroračunalniku Raspberry Pi smo postavili strežnik LAMP (angl. Linux Apache MySQL PHP). Načrtovali smo podatkovno zbirko za zbiranje podatkov iz senzorjev. Napisali smo skripto, ki podatke prejme iz vmesnika UART in jih vnese v podatkovno zbirko MySQL. Načrtovali smo spletno storitev REST, ki streže podatke iz zbirke.

V 5. poglavju je opisan postopek razvoja mobilne aplikacije. Opisali smo postopek pridobitve podatkov o vremenu iz portala Yahoo Weather - s pomočjo ustreznega vmesnika API. Posamezna podpoglavja v poglavju predstavljajo aktivnosti aplikacije. Aplikacijo smo naredili uporabniku prijazno. Razvili smo sistem procesiranja slik, ki sliko tlorisa stavbe obogati z barvami, ki predstavljajo temperaturo ali vlago. Na takšni sliki uporabnik lahko zelo hitro vidi, če so sobe na ustrezni temperaturi. Hitro se tudi vidi, če katera soba odstopa od drugih. V aplikacijo smo tudi dodali možnost ogleda zgodovine z grafi. S pomočjo grafov lahko uporabnik ugotovi, če je stavba ustrezno ogrevana oz. hlajena in se energija ne zapravlja za prazne sobe.

V 6. poglavju so predstavljeni še rezultati naloge in možne izboljšave.

1.1 Predstavitev BLE multi-hop mehanizma

Delovanje mehanizma je predstavljeno na sliki 1.1. Mehanizem je sestavljen iz posebnih vozlišč, tako imenovanih „brezžičnih multisenzorjev IJS“. Multisenzorji so na sliki predstavljeni kot N0, N1, N2, N3, N4. Multisenzorji so osnovani na komunikacijskem modulu Bluegiga BLE113, ki ima vgrajen mikrokrmilnik 8051 [10]. Strojna oprema multisenzorjev je bolj podrobno opisana v 2. poglavju. Programska oprema modula BLE113, ki omogoča multi-hop mehanizem je razvita v programskem jeziku BGScript [2].



Slika 1.1: Delovanje IJS BLE multi-hop mehanizma

Vsak brezžični multisenzor prevzame vlogo gospodarja in sužnja. To sta stanji na povezovalni plasti BLE 4.0. V stanju sužnja, BLE naprava oglašuje svoje stanje. V stanju gospodarja, BLE naprava išče sužnje in se na njih poveže. V primeru multi-hop mehanizma so vse naprave v stanju sužnja, dokler se ne zažene zajemanje podatkov z multi-hop mehanizmom. Na sliki 1.1 je primer, kjer je zajemanje podatkov sproženo na vozlišču N0.

Naprava N0 se preklopi v način gospodarja in se poveže na vse dosegljive sužnje. Ko sužnji prejmejo povezavo od gospodarja se tudi oni preklopijo v način gospodarja in rekurzivno iščejo sužnje. Ta postopek se ponavlja dokler ne pride do lista drevesne strukture, ki ni našel svojih sužnjev (N2, N3, N4). Takšen suženj (N3, N4) samo prebere podatke iz senzorjev in jih vrne svojemu gospodarju (N1). Naprava N2 stori enako, le da podatke vrne svojemu gospodarju (N0). Gospodar (N1) podatkom vseh svojih sužnjev doda še podatke iz svojih senzorjev, se preklopi nazaj v način sužnja in podatke vrne svojemu gospodarju (N0). Ta postopek se rekurzivno ponavlja, dokler podatki iz vseh odkritih naprav ne pridejo v koren drevesne strukture in se mehanizem zaključi (N0).

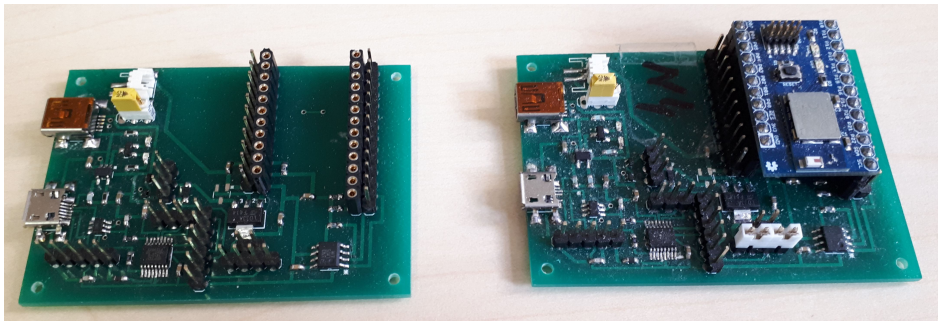
Poglavje 2

Uporabljena strojna oprema

V poglavju je predstavljena vsa uporabljena strojna oprema. To vključuje „brezžični multisenzor IJS“ in komponente ki ga sestavljajo. Za posamezne komponente je zapisano zakaj so bile izbrane. Prav tako je opisana strojna oprema mikroračunalnika Raspberry Pi, ki smo ga uporabili za zajem podatkov in spletni strežnik.

2.1 Brezžični multisenzor IJS

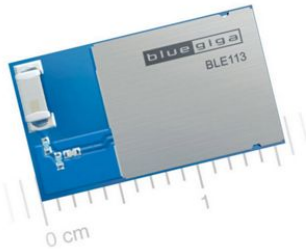
Brezžični multisenzor IJS je razvojna plošča, razvita na IJS za potrebe članka "Multi-hop komunikacija v Bluetooth Low Energy ad-hoc brezžičnem senzorskem omrežju" [10]. Izdelanih je bilo 8 plošč od tega smo jih 6 nadgradili z izboljšano programsko opremo in novimi senzorji. Razvojna plošča je prikazana na sliki 2.1. Glavna komponenta razvojne plošče je komunikacijski modul Bluegiga BLE113. Uporabljena so bila že izdelana manjša tiskana vezja s priključki, ki omogočajo priklop modula BLE113 na prototipnih vezjih. Tiskano vezje z modulom BLE113 se priklopi na tiskano vezje razvito na IJS s pomočjo priključkov. Multisenzor IJS poleg modula vsebuje tudi napajalnik, vmesnik UART preko microUSB ali miniUSB in LM75B digitalni temperaturni senzor.



Slika 2.1: Brezžični multisenzor IJS brez in z modulom BLE113

2.1.1 Bluetooth komunikacijski modul BLE113

Pametni komunikacijski modul Bluegiga BLE113 je namenjen napravam z majhno porabo električne energije. Modul BLE113 ima vgrajen Bluetooth radio, možno pa je tudi nalaganje uporabniških programov [1]. Podrobnejše specifikacije modula BLE113 so podane v tabeli 2.1. Modul BLE113 je bil uporabljen kot osrednja komponenta brezžičnih multisenzorjev IJS. Pri izbiri modula BLE, se je upoštevalo več parametrov. Modul mora biti sposoben delovati v načinu sužnja in gospodarja in pri tem porabiti čim manj energije. Modul BLE113 je v načinu gospodarja sposoben istočasno komunicirati z največ 8 sužnji, kar je več od drugih modulov s podobno porabo energije [3]. Ker modul omogoča programiranje z uporabniško kodo, je bil brezžični multisenzor IJS razvit kot samostojna BLE naprava brez zunanjega CPUja [10].



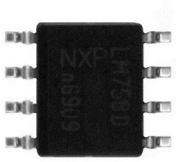
Slika 2.2: Komunikacijski modul BLE113. Vir slike: [11]

Bluetooth Smart stack	GAP, GATT, L2CAP and SMP
Moč oddajanja	0 dBm do -23 dBm
Občutljivost prejemanja	-93 dBm
Poraba ob pošiljanju	18.2 mA
Poraba ob prejemanju	14.3 mA
Poraba v načinu spanja	0.4 uA
Zunanje naprave	UART, SPI, I^2C, PWM, GPIO, 12-bit ADC

Tabela 2.1: Lastnosti modula BLE113. Vir tabele: [1]

2.1.2 Senzor temperature LM75B

Digitalni senzor temperature LM75B je prikazan na sliki 2.3. Vgrajen je na tiskanem vezju brezžičnih multisenzorjev IJS. Z modulom BLE113 komunicira preko vodila I^2C. Temperaturo preko vodila pošlje kot 11-bitno vrednost v dvojiškem komplementu. Merilno območje senzorja je od -55°C do 125°C z natančnostjo 0,125°C [4]. Med nadgradnjo, smo dodali senzor HIH6100, ki meri temperaturo in vlago, zato LM75B ni bil več potreben.

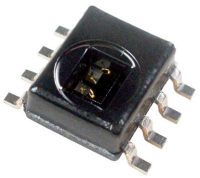


Slika 2.3: Senzor LM75B. Vir slike: [14]

2.1.3 Senzor temperature in vlage HIH6100

Digitalni senzor Honeywell HIH6130-021-001 je prikazan na sliki 2.4. Je senzor temperature in vlage s katerim smo razširili funkcionalnost brezžičnih senzorjev. Izmerjene vrednosti lahko na zahtevo pošlje preko vodila SPI ali

I^2C. Tako vlago, kot temperaturo vrne kot nepredznačeno 14-bitno vrednost, kjer najmanjša in največja vrednost predstavljata meji merilnega območja. Pri vlagi je merilno območje od 0% do 100%. Pri temperaturi je merilno območje od -40°C do 125°C [7]. Postopek pretvorbe vrednosti v ustrezne enote je opisan v 3. poglavju.



Slika 2.4: Senzor HIH6130-021-001. Vir slike: [13]

Pri izbiri novega senzorja smo upoštevali sledeče pogoje:

- digitalni senzor, pošiljanje podatkov preko SPI ali I^2C
- možnost branja temperature in vlage
- SOIC-8 ohišje.

Digitalni senzor smo potrebovali, ker ima modul BLE113 samo en vhod z možnostjo branja analogne vrednosti. Merjenje vlage in temperature v enem senzorju smo upoštevali kot pogoj zato, da ob priključitvi novega senzorja ni bilo več potrebno uporabiti starega senzorja temperature LM75B. Ohišje SOIC-8 pa je bilo izbrano zato, ker smo imeli na razpolago nekaj vmesniških SOIC-8 tiskanih vezij in je bil to najbolj praktičen način priključitve senzorjev na tiskana vezja brezžičnih senzorjev. Poleg tega, je ohišje SOIC-8 tudi razmeroma veliko in je ročno spajkanje lažje. Senzor HIH6130-021-001 je ustrezal vsem pogojem.

2.2 Mikroračunalnik Raspberry Pi

Raspberry Pi je mikroračunalnik z ARM procesorjem. Kot spletni strežnik aplikacije smo uporabili Raspberry Pi 2 model B prikazan na sliki 2.5. Za

povezavo z internetom uporablja priključek Ethernet [15]. Uporabili smo ga kot spletni strežnik s programsko opremo LAMP. Podrobnejše specifikacije računalniškega sistema so prikazane v tabeli 2.2.

Slika 2.5: Mikroračunalnik Raspberry Pi 2 model B. Vir slike: [12]

Zahteve za spletni strežnik:

- programska oprema LAMP
- internetna povezava
- vsaj 100MB za podatkovno zbirko
- vsaj 1 USB vhod za vmesnik UART in napajanje

Po specifikacijah je mikroračunalnik Raspberry Pi 2 model B več kot zadosten za potrebe spletnega strežnika in je cenovno ugoden. LAMP programska oprema za Raspberry Pi je na voljo in je prosto dostopna. Internet je dostopen preko priključka Ethernet. Podatkovno zbirko lahko z lahkoto spravimo na microSD kartico velikosti 1GB ali več. Dodatni USB vhodi so med razvojem prišli prav, saj smo jih uporabili za razhroščevanje preko vmesnika UART in za napajanje brezžičnih senzorjev.

CPU	Broadcom BCM2837 Arm7 900MHz
RAM	1GB
GPIO	40pin extended GPIO
USB	4 x USB 2 ports
Audio	4 pole Stereo output and Composite video port
Display	Full size HDMI
Network	10/100 Ethernet
Storage	Micro SD
Power	Micro USB power source

Tabela 2.2: Specifikacije mikroračunalnika Raspberry Pi 2 model B.

Poglavje 3

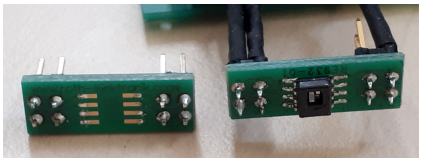
Nadgradnja brezžičnih multisenzorjev IJS

Brezžični multisenzorji IJS so bili razviti kot samostojno tiskano vezje, ki dovoljuje dodajanje novih elementov. Na tiskanem vezju se nahaja senzor LM75B, ki omogoča le branje temperature. Za potrebe aplikacije smo na vsak multisenzor IJS priključili senzor HIH6130-021-001, ki meri tako vlago, kot temperaturo. Programsko opremo brezžičnih multisenzorjev IJS je bilo potrebno prilagoditi, da bere meritve iz novega senzorja in zna preko multi-hop mehanizma BLE poslati več vrst podatkov.

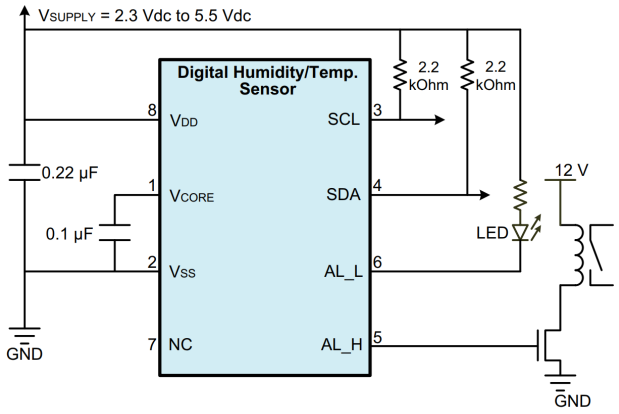
3.1 Priključitev senzorja temperature in vlage

Senzorje je bilo najprej potrebno prispajkati na vmesniška tiskana vezja kot je prikazano na sliki 3.1. Pri spajkanju senzorjev je bilo potrebno paziti, da se ne prekorači 350 °C za več kot 4s, kot je navedeno v podatkovni listini [7]. Ko so bili senzorji prispajkani, smo jih na brezžične senzorje priklopili po vezavi prikazani na sliki 3.2.

Od vseh elektronskih elementov naštetih na sliki 3.2 je bilo potrebno dodati samo en 0.1μF kondenzator. Upor na Vdd ni bil potreben, prav tako ni bilo potrebno dodati pull up uporov na priključka SDA in SCL, saj so že



Slika 3.1: SOIC-8 tiskano vezje brez in s senzorjem.

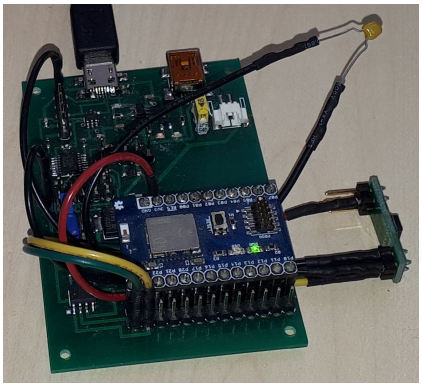


Slika 3.2: I²C vezava za senzor HIH6130-021-001. Vir slike: [7]

vgrajeni na tiskanem vezju. Na razvojni plošči so bile vse povezave že na voljo v obliki priključkov, zato ni bilo potrebno uporabiti dodatnih razvojnih plošč. Izgled brezžičnega multisenzorja IJS po priklopu novega senzorja vlage je prikazan v sliki 3.3.

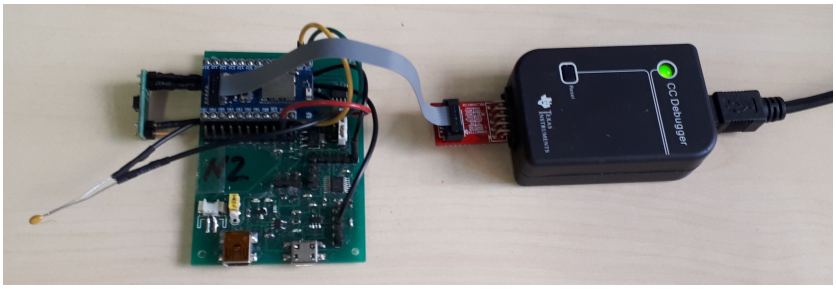
3.2 Vključitev novega senzorja v programsko opremo

Za programiranje modula BLE113 se uporablja programski jezik BGScript. BGScript podpira programiranje z dogodki kot so prekinitve in Bluetooth dogodki. Njegov vmesnik API vsebuje funkcije za delo z Bluetoothom in funkcije za delo z UART, USB, SPI, I²C, GPIO, PWM in ADC [2].



Slika 3.3: brezžični senzor s priklopljenim senzorjem HIH6130-021-001.

Za programiranje modulov BLE113 smo uporabili Bluegiga BLE SW Update Tool v1.3.6, ki prevede BGScript v strojno kodo primerno za nalaganje. Za nalaganje strojne kode na modul BLE113 smo uporabili CC Debugger viden na sliki 3.4.



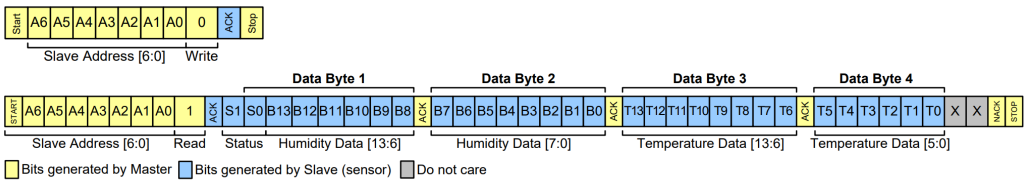
Slika 3.4: Razvojna plošča (levo) in CC Debugger (desno).

Ko je bil senzor HIH6130-021-001 povezan je bilo potrebno v vgrajeni programski opremi modula BLE113 narediti dve spremembi:

- prejemanje podatkov preko vodila I^2C za nov senzor
- pravilno posredovanje večih tipov podatkov med brezžičnimi senzorji

3.2.1 Prejemanje podatkov preko vodila I²C

Kot je zapisano v podatkovni listini senzorja HIH6130-021-001, se branje podatkov iz vodila izvede v dveh korakih. Prvi korak je „Measurement Request“ prikazan na sliki 3.5 zgoraj. S prejetjem tega ukaza se senzor zbudi in prične z merjenjem veličin, kar tipično traja 37 ms [6]. Za tem se lahko pošlje „Data Fetch“ in senzor vrne podatke kot je prikazano na sliki 3.5 spodaj. Privzeti I²C naslov senzorja je 0x27 [6].



Slika 3.5: Format podatkov na vodilu I²C. Vir slike: [6]

Spodaj je navedena BGScript koda za ukaza „Measurement Request“ in „Data Fetch“.

```
#measurement request
call hardware_i2c_write(($27<<1),1,0,"\x00")
                        (data_i2c_len)
#data fetch
call hardware_i2c_read(($27<<1), 1, 4)
                       (i2c_result, data_i2c_len, data_i2c(0:4))
```

Od vsakega podatka smo uporabili samo 8 najpomembnejših bitov oziroma MSB, saj mehanizem multi-hop vrednosti pošilja v 8 bitni obliki. Za pretvarjanje smo uporabili izraza (3.1) in (3.2) iz podatkovne listine [6].

$$Humidity(\%RH) = \frac{Humidity\ Output\ Count}{(2^{14} - 2)} \cdot 100\% \tag{3.1}$$

$$Temperature(^{\circ}C) = \frac{Temperature\ Output\ Count}{(2^{14} - 2)} \cdot 165 - 40 \tag{3.2}$$

Spodaj pa je še navedena BGScript koda, ki uporabi izraza (3.1) in (3.2), da pretvori prejete podatke v ustrezne enote in jih shrani v spremenljivki „data_hum“ in „data_temp“.

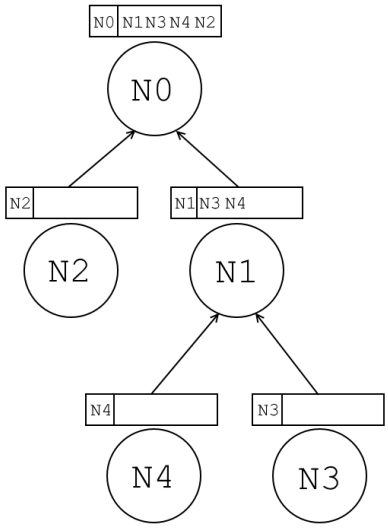
```
data_hum = (data_i2c(0:1)<<8) | data_i2c(1:1)
data_hum = data_hum & 16383 #remove first 2 bits (Status bits)
data_hum = data_hum >> 6
data_temp = (data_i2c(2:1)<<6) | (data_i2c(3:1) >> 2)
data_temp = data_temp >> 6
data_temp = ((data_temp*165)/255) - 40
data_hum = ((data_hum*100)/255)
```

V programskem jeziku BGScript s konstruktom (X:Y) izluščimo podniz danega niza. X je začetni indeks podniza Y pa njegova dolžina.

3.2.2 Prenašanje več tipov podatkov

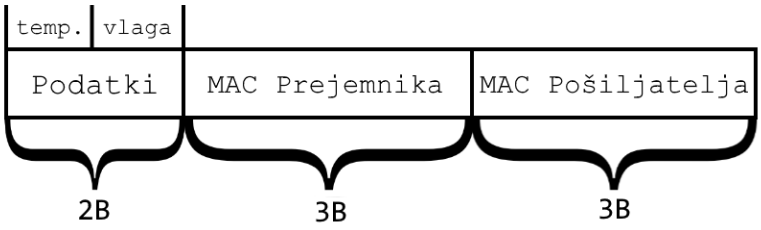
Standard BLE 4.0 sam po sebi ne podpira multi-hop mehanizma. Na IJS so razvili mehanizem za BLE, ki je predstavljen v članku [9]. Originalen mehanizem je bil namenjen samo prenosu enega tipa podatkov (temperature), zato smo ga razširili, da omogoča tudi prenos podatkov vlage. Multi-hop mehanizem je implementiran tako, da koren najprej pošlje poizvedbe za podatke, ko pa poizvedba pride do listov se podatki prenašajo nazaj navzgor. Vsako vozlišče svoje podatke doda na začetek seznama podatkov in seznam pošlje naprej, kot je prikazano na sliki 3.6. V praksi vsako vozlišče predstavlja brezžični senzor.

Format strukture podatkov, ki jo vsako vozlišče ustvari in doda na seznam podatkov je predstavljen na sliki 3.7. V stari implementaciji je vsako vozlišče ustvarilo 7 bajtno strukturo [10]. Od tega so se podatki temperature nahajali v prvem bajtu, ostalih 6 bajtov pa predstavlja zadnja konca dveh MAC naslovov. Prvi MAC pove komu se paket pošilja, drugi pa kdo ga pošilja. S pomočjo teh podatkov je v korenu mogoče rekonstruirati pot podatkov.



Slika 3.6: Prenos podatkov med brezžičnimi senzorji.

Strukturo smo razširili tako, da podatki zavzamejo 2 bajta. Od tega je prvi bajt za temperaturo, drugi pa za vlago. Če bi za temperaturo uporabili več bitov, bi sicer lahko dobili bolj natančne meritve temperature, vendar bi potem bilo še bolj omejeno največje število vozlišč v sistemu, zaradi omejenega notranjega pomnilnika modula BLE113.



Slika 3.7: Struktura podatkov, ki jih ustvari vsak brezžični senzor.

Za to spremembo smo razširili spremenljivko „node_data“ iz 7 na 8 bajtov. Spremenljivka se v kodi pogosto uporablja, zato smo pazljivo preverili vsako

uporabo in zagotovili, da se ob vsaki uporabi prenese vseh 8 bajtov. Po branju temperature iz senzorja se spremenljivka napolni, kot je prikazano v spodnjem izseku BGScript kode.

```
node_data(0:1) = data_temp
node_data(1:1) = data_hum
node_data(2:3) = adv_data(6:3)
node_data(5:3) = mac_addr(0:3)
```

3.3 Odprava pomanjkljivosti v programski opremi

V obstoječi vgrajeni programski opremi smo naleteli na pomanjkljivost pri delovanju, če je med poizvedbo kakšen brezžični multisenzor postal nedosegljiv. V originalni programski opremi se je to rešilo s konstantnim timeoutom, ki je bil za vse ravni drevesne strukture enak. Za pravilno delovanje je bilo potrebno med vozlišči pošiljati podatek o trenutni globini. Vpeljali smo novo spremenljivko *node_depth*. Spremenljivka se od gospodarja pošlje na sužnja povečana za 1. Spremenljivka *node_depth* se na sužnje pošilja znotraj dogodka *connection_status*, sužnji pa globino prejmejo znotraj dogodka *attributes_value*.

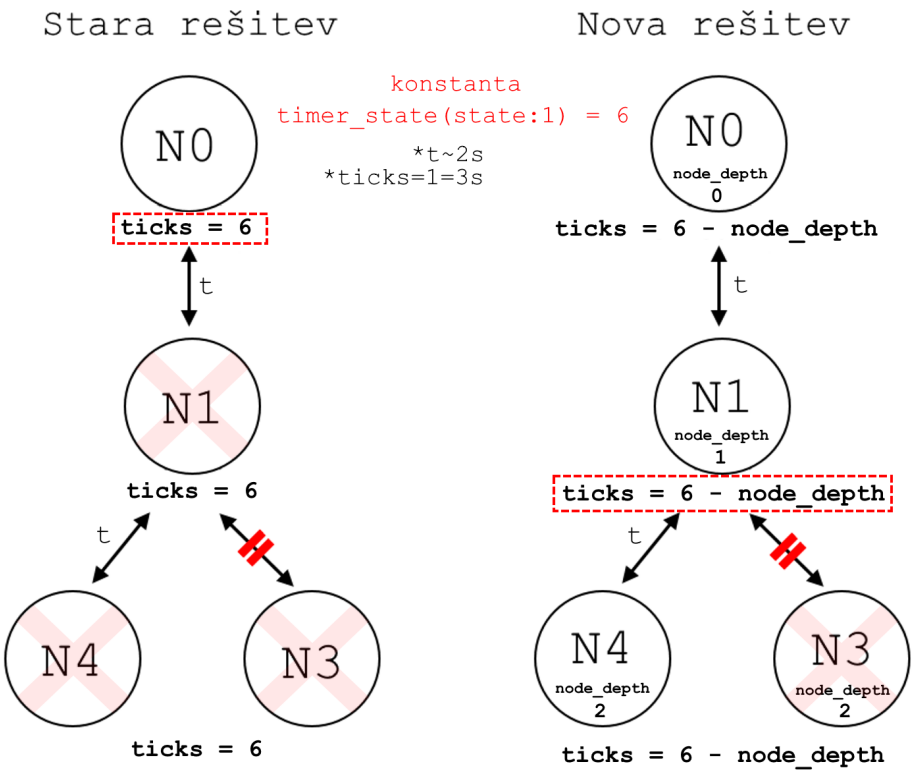
Spremembe znotraj dogodka *connection_status*:

```
...
wr_data(7:1) = node_depth + 1
...
call attclient_attribute_write(0, att_hnd_val, 8, wr_data(0:8))
...
```

Spremembe znotraj dogodka *attributes_value*:

```
event attributes_value(connection, reason, handle, offset, len, value)
...
node_depth = value(7:1)
...
```

Razlika med staro in novo rešitvijo je prikazana na sliki 3.8. Na sliki je primer, kjer je povezava z vozliščem N3 izgubljena med delovanjem. Timeout, ki se sproži je obkrožen z rdečo. Pri stari rešitvi se timeout sproži pri vozlišču N0, ostali podatki pa se izgubijo. Do izgube podatkov pride, ker vozlišče N1 še vedno čaka na podatke N3, ko se v vozlišču N0 zgodi timeout. Pri novi rešitvi se timeout pravilno sproži pri vozlišču N1, izgubijo pa se le podatki vozlišča N3. S spremenljivko t je na sliki označeno trajanje vzpostavitve povezave med vozlišči.



Slika 3.8: Stara in nova rešitev nedosegljivih vozlišč s timeouti.

Multi-hop BLE mehanizem IJS za delovanje uporablja končni avtomat. Vsako stanje končnega avtomata ima svoj timeout, ki se nastavi ob preklopu stanj. Problematično je bilo stanje *READ*. V tem stanju je multisenzor v

načinu gospodarja in čaka na odgovor svojih sužnjev. To je stanje, katerega timeouti so prikazani na sliki 3.8. Mehanizem za preklop med različnimi stanji uporablja dogodek *change_state*. Znotraj dogodka se nastavijo vsi parametri, ki so potrebni za tisto stanje, med drugim timeout. Naredili smo spremembo za dve stanji kot je prikazano spodaj:

```
#IDLE
...
node_depth = 0
...
#READ
...
ticks = timer_state(state:1) - node_depth
...
```

Spodaj je razložen pomen spremenljivk:

- *ticks* predstavlja timeout za stanje
- *timer_state* je tabela timeoutov za podano stanje
- *state* je številka trenutnega stanja

Vrednost v tabeli *timer_state* za *READ* je 6. Ukaz *timer_state(state:1)* torej iz tabele prebere 1 bajt za stanje *READ*. Ena vrednost v spremenljivki *ticks* predstavlja približno 3 sekunde. Vsaka dodatna globinska raven v multihop mehanizmu, mehanizem podaljša za nekaj več kot 2 sekundi (t). Ker zmanjšanje timeouta za 3 sekunde na globinsko raven dobro ustreza, je bilo dovolj od začetnega timeouta odšteti kar spremenljivko *node_depth*. Timeout za stanje *READ* začnemo pri 18 sekundah in ga z vsako globinsko ravnjo zmanjšamo za 3 sekunde. Ker s tako rešitvijo pri globini 6 vozlišč dosežemo timeout 0, je to največja možna globina.

Poglavje 4

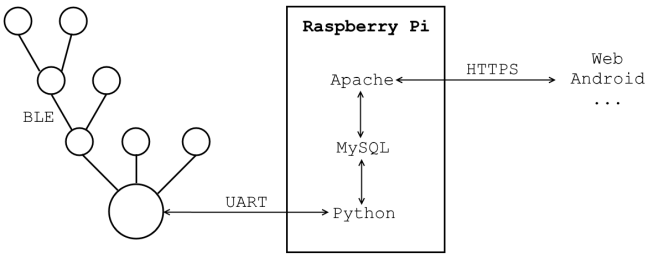
Postavitev spletnega strežnika

Kot spletni strežnik in podatkovno zbirko meritev smo uporabili mikroračunalnik Raspberry Pi 2 Model B. Kot operacijski sistem smo namestili Raspbian Stretch. Raspbian je osnovan na debianu, zato je možno nameščati programsko opremo iz repozitorija [22]. Za spletni strežnik smo uporabili programsko opremo Apache in PHP7, za podatkovno zbirko pa MySQL.

Spletni strežnik je preko vmesnika USB/UART povezan na korenski brezžični senzor. Strežnik senzorju periodično pošlje ukaz za zajemanje meritev preko vmesnika UART. Podatki preko multi-hop mehanizma od listov pridejo nazaj v korenski senzor, ki jih posreduje strežniku preko vmesnika UART. Strežnik ustrezno pretvori podatke in jih shrani v podatkovno zbirko MySQL. Podatki iz podatkovne zbirke MySQL so na zunaj dostopni preko storitve REST. Delovanje strežnika je prikazano na sliki 4.1.

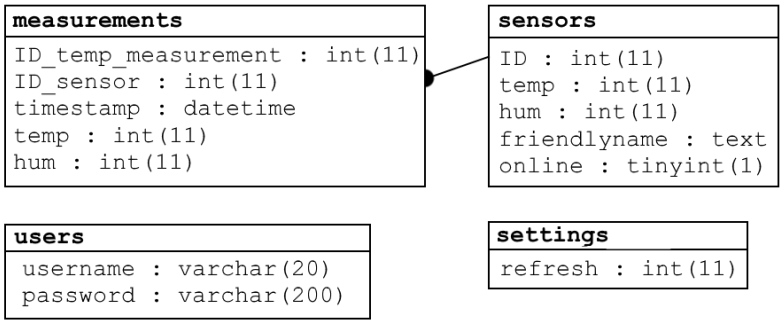
4.1 Podatkovna zbirka

Podatkovna zbirka vsebuje 4 tabele. Podatkovni model je viden na sliki 4.2. Tabela „sensors“ je tabela brezžičnih senzorjev. Ključ tabele predstavlja MAC naslov vsakega brezžičnega senzorja. Ko je senzor prvič zaznan, se ustrezen vnos samodejno ustvari. V tabeli je shranjeno tudi uporabniško poimenovanje senzorja, zadnje prebrane vrednosti meritev in podatek o tem, če



Slika 4.1: Delovanje spletnega strežnika.

so bile meritve senzorja prebrane pri zadnjem zajemanju. Tabela „measurements“ vsebuje podatke, čas in ID senzorja meritve. Tabela „users“ vsebuje uporabniško ime in zgoščeno vrednost gesla SHA-1. Tabela „settings“ vsebuje samo nastavitev refresh, ki Python skripti pove v kakšnih intervalih naj zajema.



Slika 4.2: Podatkovni model podatkovne zbirke „smarthome“.

Za dostop do podatkovne zbirke „smarthome“ smo ustvarili MySQL uporabnika „smarthome“, ki ima pravice za dostop le do zbirke „smarthome“. Sledijo parametri s katerimi se je možno lokalno povezati v podatkovno zbirko npr. z uporabo Pythona ali PHPja.

```
hostname = "localhost"
username = "smarthome"
password = "iotsmartpass"
database = "smarthome"
```

4.2 Skripta za zajemanje podatkov

Skripto za zajemanje podatkov smo napisali v programskem jeziku Python. Ob zagonu operacijskega sistema se skripta samodejno požene. Naloge skripte so:

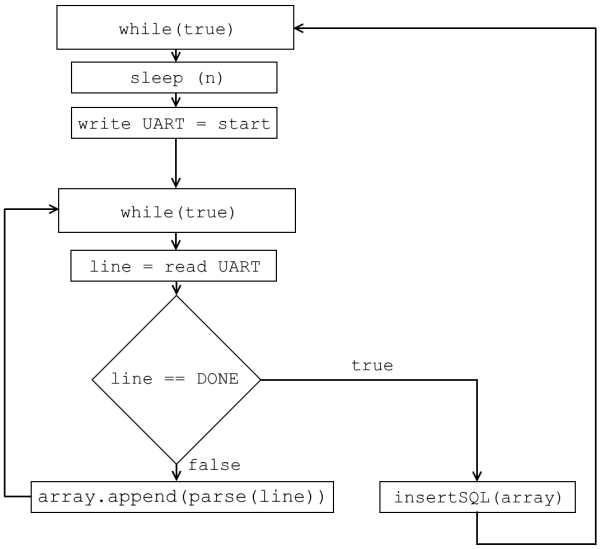
- periodično proženje poizvedbe po merjenih podatkih s senzorjev
- vnos podatkov v podatkovno zbirko MySQL
- preverjanje dostopnosti brezžičnih senzorjev

4.2.1 Branje iz vmesnika UART

Skripta izvrši branje iz senzorjev vsakih n sekund, kjer je podatek n shranjen v podatkovni zbirki. Skripta najprej preko vmesnika USB/UART pošlje ukaz za branje kot je prikazano na sliki 4.3. Branje senzorjev tipično traja 10–20 sekund odvisno od števila in globine senzorjev. Podatki se najprej zberejo v začasni tabeli, ko pa je branje končano, se prenesejo v podatkovno zbirko.

4.2.2 Shranjevanje v podatkovno zbirko

Ko skripta prejme podatke iz senzorjev, jih shrani v podatkovno zbirko. Najprej preveri, če je prebrani senzor že v tabeli „sensors“, v nasprotnem primeru ga ustvari. Za vsak zaznan senzor posodobi njegov vnos v tabeli „sensors“ in ustvari nov vnos v „measurements“ s trenutnim časovnim žigom. Stolpec „online“ se za senzorje, ki so bili zaznani ob zadnjem zajemanju postavi na 1, za preostale pa na 0. Podrobnejše delovanje vnašanja v podatkovno zbirko je prikazano na sliki 4.4. Za povezavo Pythona s podatkovno zbirko smo



Slika 4.3: Delovanje Python skripte za zajemanje podatkov.

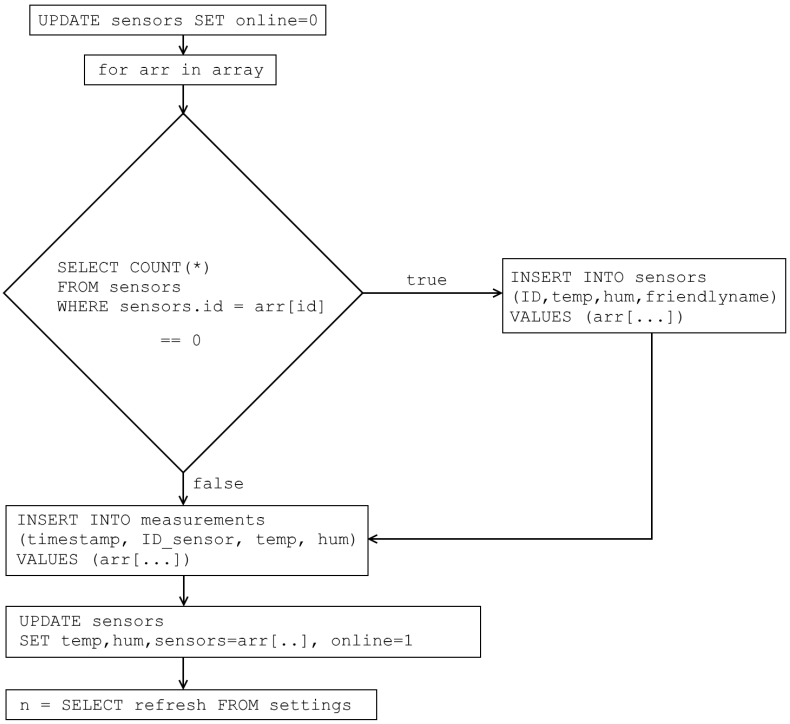
uporabili paket „python-mysqldb“. Spodaj je prikazan postopek povezave s podatkovno zbirko.

```
db = MySQLdb.connect(hostname, username, password, database)
cur = db.cursor()
cur.execute("UPDATE sensors SET `online`=0")
db.commit()
```

4.3 Spletna storitev

Za potrebe Android aplikacije smo ustvarili spletno storitev, ki podatke iz podatkovne zbirke MySQL vrača v JSON formatu. Za razvoj spletne storitve smo uporabili programski jezik PHP. V tabeli 4.1 so predstavljeni vsi možni API klici, parametri, ki jih potrebuje in kaj vračajo. Vsak izmed klicev je PHP datoteka na strežniku, do katere je možen dostop preko url naslova npr.:

```
https://127.0.0.1/getcurrent.php
```



Slika 4.4: Vnašanje podatkov v podatkovno zbirko MySQL.

Spodaj je razlaga API klicev predstavljenih v tabeli 4.1:

- changepassword - spremeni geslo za podanega uporabnika
- getcurrent - vse senzorje in njihovo zadnje stanje v JSON tabeli
- getdailyavg - vrne dnevna povprečja za senzor v JSON tabeli
- gethourlyavg - vrne povprečja po urah v JSON tabeli
- gethistory - vrne vsa branja senzorja v JSON tabeli
- setrefreshtime - nastavi čas med branji
- setsensorname - nastavi uporabniško ime senzorja
- login - vrne napako, če uporabniško ime in geslo ni pravo
- delete - izbriše senzor in vse meritve iz podatkovne zbirke

Storitev	**Telo**	**Vrača**
changepassword	old, new, user	"OK", 400, 401
getcurrent	username, password	JSON tabela, 401
getdailyavg	username, password, id	JSON tabela, 400, 401
gethourlyavg	username, password, id	JSON tabela, 400, 401
gethistory	username, password, id	JSON tabela, 400, 401
setrefreshtime	username, password, refresh, type	JSON objekt, "OK", 400, 401
setsensorname	username, password, id, name	"OK", 400, 401
login	username, password	"OK", 400, 401
delete	username, password, id	"OK", 400, 401

Tabela 4.1: Tehnične specifikacije spletne storitve.

Primer JSON tabele za klic *https://127.0.0.1/getcurrent.php*.


```
[
    {
        "nodeID": "3746845",
        "friendlyname": "N5",
        "temp": "27",
        "hum": "36",
        "online": "0",
        "timestamp": "2018-06-20 12:17:31" },
    {
        "nodeID": "3746855",
        "friendlyname": "N2",
        "temp": "27",
        "hum": "36",
        "online": "1",
        "timestamp": "2018-06-20 12:17:31" }
]
```

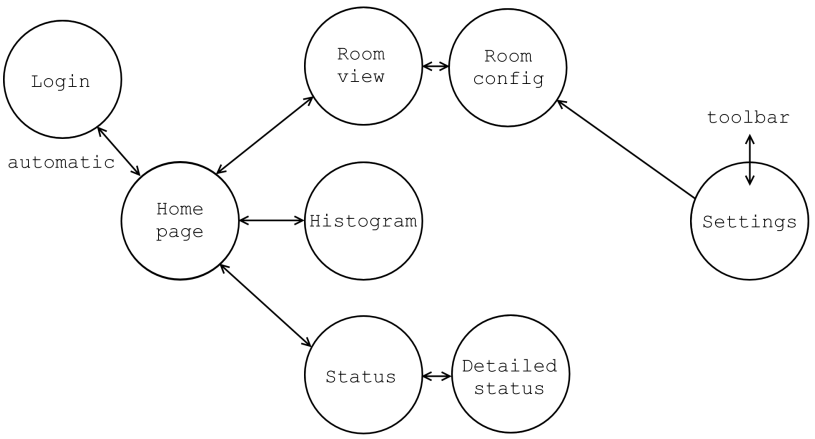
Primer JSON tabele za klic *https://127.0.0.1/getdailyavg.php*


```
[
    {
        "timestamp": "2018-03-08 08:58:51",
        "temp": "25",
        "hum": "58" },
    {
        "timestamp": "2018-03-09 00:04:04",
        "temp": "24",
        "hum": "59" }
]
```

Poglavje 5

Razvoj mobilne aplikacije

Mobilno aplikacijo smo razvili za operacijski sistem Android z uporabo okolja Android Studio. Za programiranje smo uporabili programski jezik Java. Aplikacija za delovanje potrebuje mrežni dostop do strežnika Raspberry Pi, za nekatere opcijske funkcionalnosti pa tudi dostop do interneta. Vse aktivnosti aplikacije in kako uporabnik prehaja med njimi so prikazane na sliki 5.1.



Slika 5.1: Shema prehodov med aktivnostmi aplikacije.

Ko uporabnik prvič zažene aplikacijo se prikaže vodič za konfiguracijo in prijavo, kar je podrobneje opisano v poglavju 5.1. Ko je prva konfiguracija zaključena, se odpre glavni meni prikazan na sliki 5.2. Glavni meni vsebuje

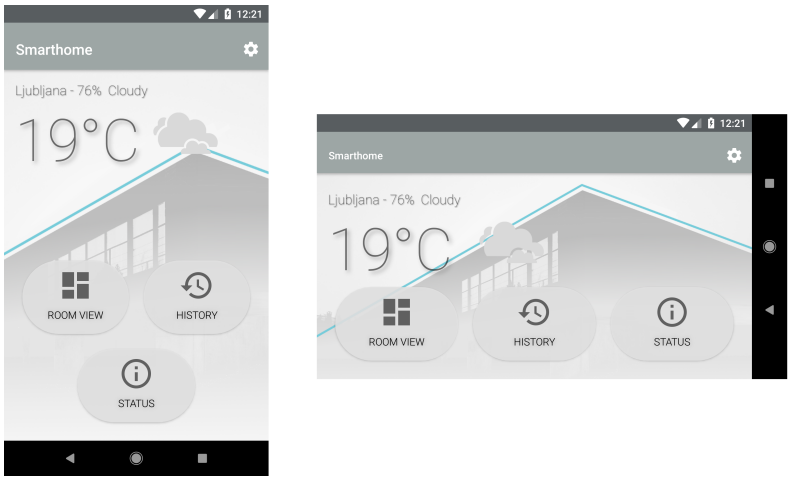
povezave na glavne tri funkcije aplikacije:

- pregled sob
- zgodovina
- status

Poleg teh treh gumbov je v glavnem meniju prikazano tudi trenutno vreme v izbrani lokaciji. Prikazane so informacije o temperaturi in vlagi, stanje vremena pa je prikazano z ikono. Za informacije o vremenu smo uporabili programski vmesnik Yahoo Weather API, ki je podrobneje opisan v poglavju 5.2.

Na vrhu zaslona je vedno na voljo tudi statusna vrstica, ki prikazuje naslov trenutne aktivnosti in povezavo v aktivnost „nastavitve“. Glede na trenutno aktivnost se v statusni vrstici prikažejo različne funkcije. Gumb za nastavitve je vedno prisoten, gumb nazaj pa je prisoten v vseh aktivnostih razen v glavnem meniju. Statusna vrstica v nekaterih aktivnostih prikazuje spustni seznam za izbiro elementov.

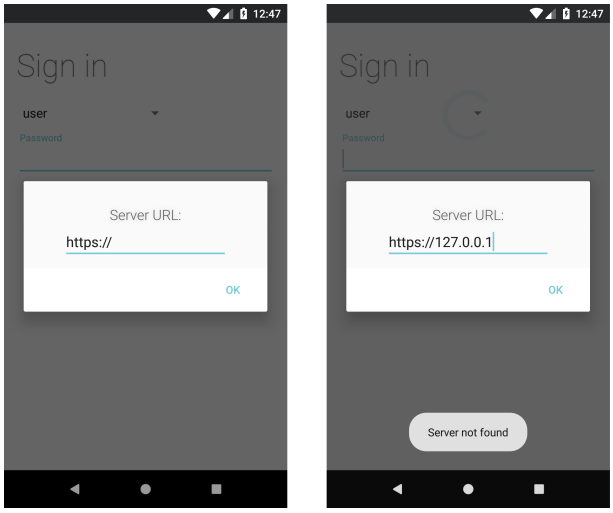
Kot je razvidno iz slike 5.2, aplikacija zagotavlja podporo tako vertikalni kot horizontalni postavitvi naprave. Zagotovljena je tudi podpora napravam z različno gostoto točk DPI (tablice, mobilni telefoni, ...)



Slika 5.2: Glavni meni aplikacije.

5.1 Prijava

Ker je spletna storitev javno dostopna v internetu, smo zagotovili nekaj varnosti. Za prenos podatkov se uporablja protokol HTTPS s samo–podpisanim certifikatom. Dostop do storitev pa je omejen z uporabniškim imenom in geslom. Ob vsakem zagonu aplikacija preveri, če so v notranjem pomnilniku shranjeni podatki o uporabniku in strežniku. Če ne, se zažene vodič za konfiguracijo. Vodič za konfiguracijo najprej odpre pojavno okno, kjer je potrebno vnesti naslov strežnika. Če je vneseni naslov napačen ali ni dostopeen, se prikaže napaka kot je prikazano na sliki 5.3. Za te vrste napak smo uporabljali gradnik „toast“, ki je primeren gradnik za kratka, neinteraktivna sporočila [18].

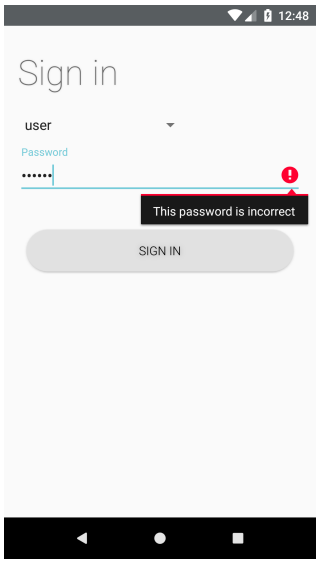


Slika 5.3: Vnos naslova strežnika.

Ko je naslov strežnika ustrezno vnesen, je omogočena prijava. Za potrebe aplikacije smo uporabili samo dva uporabnika – „user“ in „admin“. Zato je kot vnosno polje uporabniškega imena uporabljen spustni seznam. Ob vnosu gesla v polje so črke zakrite, ker je uporabljen gradnik za gesla. Za preverjanje uporabniškega imena in gesla ob prijavi je uporabljen API klic „login“ opisan v poglavju 4.3. Če je vneseno geslo napačno, vmesnik API

vrne HTTP statusno kodo 401. Ob tej kodi se ob gradniku za geslo pojavi opozorilo kot je prikazano na sliki 5.4. Za to vrsto izpisa napake je potrebno le klicati metodo „seterror“ objekta „edittext“, kot je prikazano v kodi spodaj.

```
mPasswordView.setError(getString("Napačno geslo"));
mPasswordView.requestFocus();
```



Slika 5.4: Prijava uporabnika in izpis napake.

5.2 Pridobivanje podatkov o vremenu

Informacije o vremenu, trenutni zunanji temperaturi in vlagi so bile potrebne na večih mestih v aplikaciji. Med drugim za prikaz v glavnem meniju in v aktivnosti „pregled sob“. Ker so brezžični senzorji namenjeni notranji uporabi, smo za pridobivanje informacij o vremenu uporabili programski vmesnik Yahoo Weather API. Posebnost programskega vmesnika je, da za poizvedbe uporablja poseben povpraševalni jezik YQL podoben SQL. Programski vmesnik Yahoo Weather API omogoča REST poizvedbe.

5.2.1 Programski vmesnik Yahoo Weather API

Primer YQL, ki vrne vse podatke o vremenu za Ljubljano in za temperaturo uporablja stopinje Celzija.

```
select * from weather.forecast where woeid in
(select woeid from geo.places(1) where text='ljubljana')
and u='c'
```

V aplikaciji smo uporabili enak YQL, le da smo niz Ljubljana nadomestili s podatki shranjenimi v notranjem pomnilniku aplikacije. Uporabnik tako lahko v nastavitvah sam vnese mesto za katerega bo prikazano vreme. Spodaj je naveden še uporabljeni klic za Yahoo Weather API. Spremenljivka „searchtext“ vsebuje zgoraj navedeni YQL, formatiran v primernem formatu za URL.

```
https://query.yahooapis.com/v1/public/yql?q="+searchtext+"
                                            &format=json
```

Vsakič ko uporabnik odpre glavno stran, aplikacija preveri če so podatki o vremenu starejši od 6 minut. V primeru da so starejši, se posodobijo. Uporabnik lahko tudi ročno posodobi podatke z gesto „poteg navzdol“. V ta namen smo uporabili gradnik „swipe layout“, ki zazna poteg navzdol in tudi prikaže vrtečo puščico.

5.2.2 Razred AsyncTask

Posodabljanje vremena, kot vse druge API klice v aplikaciji smo izvedli z uporabo razširjenega AsyncTask razreda. To omogoča gladko izvajanje niti uporabniškega vmesnika, med tem, ko v ozadju potekajo dalj trajajoča dela [16]. V spodnjem primeru je razred „Async“ razširitev razreda „AsyncTask“, url pa predstavlja vhodni parameter. Ob spodnjem klicu se izvede notranja metoda „doInBackground“, kamor smo dodali ustrezno kodo npr. iz knjižnice Volley, ki je opisana v naslednjem razdelku.

```
new Async().execute(url);
```

5.2.3 Knjižnica Volley

Za API klice smo uporabili knjižnico Volley. Je Googlova prosto dostopna knjižnica, ki implementira protokol HTTP [20]. Kot priporoča Google, smo za implementacijo vrste RequestQueue uporabili razred edinec MySingleton(). Za implementacijo knjižnice Volley znotraj orodja Android Studio, je bilo potrebno dodati eno vrstico v datoteko build.gradle, kot je navedeno spodaj.

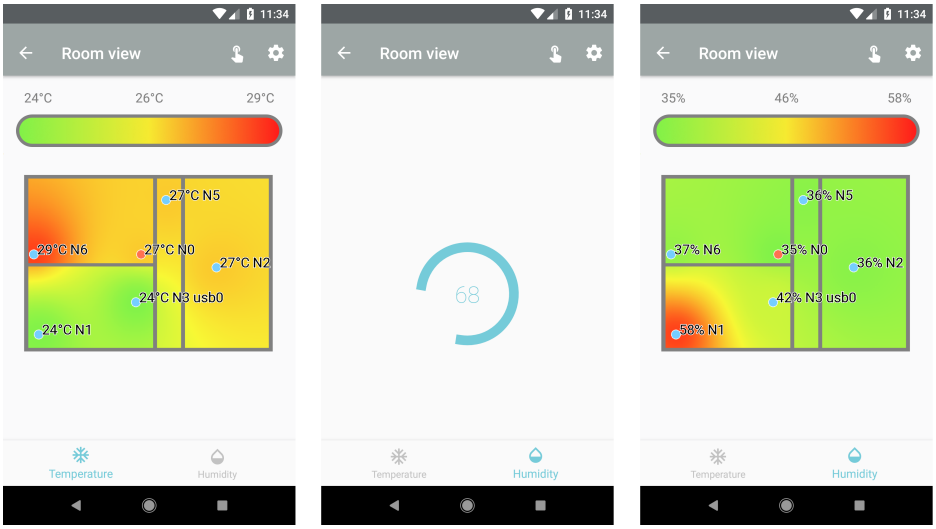
```
dependencies {
    ...
    implementation 'com.android.volley:volley:1.1.0'
}
```

5.3 Pregled sob

Aktivnost „pregled sob“ ali „room view“ je namenjena vizualni predstavitvi stanja senzorjev. Za prikaz stanja je uporabljen tloris stavbe. Uporabnik lahko tloris poljubno prilagodi, tako je aplikacija primerna za katero koli stavbo. Prav tako lahko uporabnik prilagodi lokacijo posameznih brezžičnih senzorjev znotraj tlorisa. Temperatura in vlaga sta v tlorisu predstavljena z barvo. Barva med posameznimi senzorji gladko prehaja, razen, če je na poti stena. Podatki temperature in vlage se osvežijo z vsakim vstopom v aktivnost. Podatki za temperaturo in vlago izven stavbe so pridobljeni iz programskega vmesnika Yahoo Weather API.

Aktivnost je sestavljena iz 4 komponent:

- statusne vrstice
- legende
- procesirane slike
- zavihka

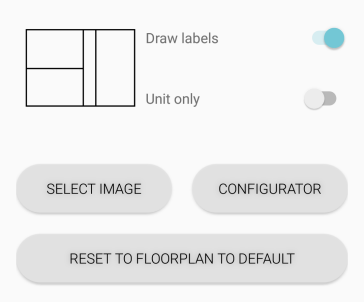


Slika 5.5: Aktivnost „room view“ v različnih stanjih.

V statusni vrstici je povezava v aktivnost „room configuration“, kjer je možna konfiguracija pozicij senzorjev. Konfiguracija pozicij senzorjev je bolj natančno opisana v poglavju 5.3.1. Legenda prikaže povezavo med barvo in prikazano temperaturo oz. vlago. Legenda je lahko avtomatska, lahko pa si jo po meri prilagodi uporabnik. Procesiranje slike tipično traja 10–20 sekund, zato smo dodali prikaz napredka v odstotkih, kot je prikazano na sliki 5.5. Aktivni senzorji so prikazani z modro piko, neaktivni pa z rdečo. Zavihka na spodnjem delu aktivnosti sta namenjana preklopu med pogledom vlage in pogledom temperature. Na levi strani slike 5.5 je pogled temperature, na desni pa pogled vlage. Trenutno izbrani zavihek je označen z modro barvo.

Določitev tlorisa je možna v nastavitvah, kot je prikazano na sliki 5.6. Za uporabniško določanje tlorisa smo uporabili rastrsko sliko. Sliko je možno naložiti iz slikovne datoteke iz pomnilnika naprave. Za izbiranje slike nismo razvili lastnega urejevalnika datotek, temveč smo uporabili razred „intent“ s sledečimi parametri:

```
new Intent(Intent.ACTION_PICK,
android.provider.MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
```



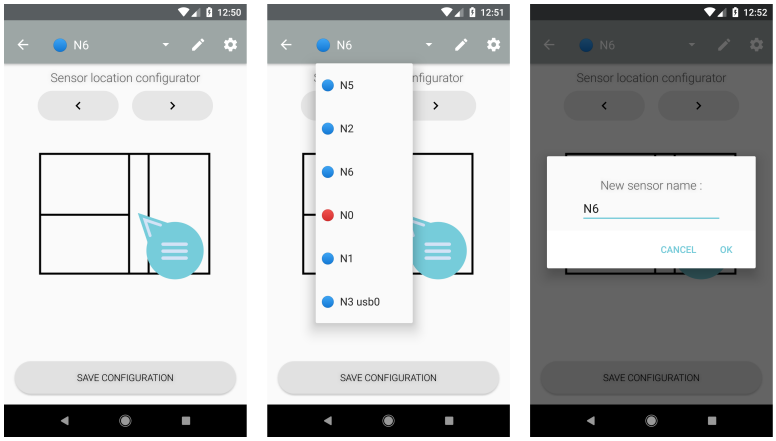
Slika 5.6: Nastavitve, ki se nanašajo na določitev tlorisa.

Takšen razred „intent“ dovoljuje uporabo aplikacije, ki že obstaja na uporabnikovi napravi za izbiro slike. Ko uporabnik v drugi aplikaciji izbere sliko, se URI slike vrne v razredu „OnActivityResult“, kot del povratnega „intenta“ iz aplikacije za izbiro slike. Da ni težav z Android sistemom dovoljenj, smo sliko takoj po vračilu shranili v notranji pomnilnik aplikacije.

V aplikacijo je vgrajena privzeta slika, ki je prikazana na sliki 5.6. Ločljivost in razmerje lastne slike sta poljubna. Slika mora imeti enobarvno ozadje, ločilne stene pa morajo biti iz enake barve. Sliko je priporočljivo shraniti v brezizgubnem formatu PNG, saj pri shranjevanju v izgubni kompresiji nastanejo napake, ki bi bile vidne na procesirani sliki.

5.3.1 Konfiguracija pozicij senzorjev

V aktivnosti „konfiguracija senzorjev“ je možno premikanje brezžičnih senzorjev po tlorisu. V statusni vrstici je spustni seznam, na katerem je prikazan trenutno izbrani senzor. Aktivni senzorji imajo na spustnem seznamu poleg imena modro piko, neaktivni pa rdečo. Poleg izbire senzorja iz spustnega seznama, se je med senzorji možno pomikati tudi z gumboma naprej in nazaj. V statusni vrstici je tudi gumb za preimenovanje senzorja. S pritiskom na gumb za preimenovanje se odpre pojavno okno s poljem za preimenovanje trenutno izbranega senzorja v spustnem seznamu, kot je prikazano na sliki 5.7.



Slika 5.7: Aktivnost „sensor configuration“ v različnih stanjih.

Za določanje pozicije senzorja smo uporabili kazalec viden na levi strani slike 5.7. Konec puščice kazalca predstavlja lokacijo senzorja na procesirani sliki. Za takšno obliko smo se odločili, da lahko uporabnik uporabi prst a vseeno vidi kam kaže kazalec. Ob izbiri drugega senzorja se izvrši animacija, ki kazalec gladko premakne na novo pozicijo.

5.3.2 Procesiranje slike

Za prikaz in procesiranje slike delamo z grafiko nad sliko tlorisa, ki jo nastavi uporabnik. Koraki algoritma za procesiranje slike so naslednji (v oklepajih je naveden tudi relativni delež porabljenega časa):

1. pridobivanje podatkov o senzorjih iz strežnika (5%)
2. *flood fill* nad točko (0,0) loči zunanji del od notranjega (15%)
3. *flood fill* nad vsemi notranjimi točkami loči sobe med sabo (26%)
4. izračun barve za senzorje glede na legendo ($< 1\%$)
5. barvanje točk med senzorji (53%)
6. risanje točk in besedila ($< 1\%$)

Najprej se podatki pridobijo iz strežnika z uporabo programskega vmesnika API. Nato se algoritem sprehodi čez vse točke v sliki. Z uporabo algoritma *flood fill* se vsak prostor pobarva na svojo unikatno barvo. To je izvedeno tako, da se *flood fill* kliče zaporedno nad vsako točko na sliki, kjer se preskoči točke, ki jih je *flood fill* že pobarval. Pri tem se za vsako sobo kliče *flood fill* samo enkrat. Tako po barvi dveh točk lahko ločimo med prostori.

Za barvanje posameznih točk smo uporabili barvni prostor HSV (angl. Hue Saturation Value), saj povečevanje in zmanjševanje le komponente H omogoča prelivanje med barvami. Ker smo želeli samo prelivanje barve od zelene do rdeče, smo minimum komponente H nastavili na 0, maksimum pa na 90. V primeru, da je uporabnik izbral avtomatsko prilagajanje legende, se med senzorji poiščeta najmanjša in največja vrednost x_{min} in x_{max} ter nastavita na 0 in 90.

$$H_{min} = 0 \tag{5.1}$$

$$H_{max} = 90 \tag{5.2}$$

$$H(x_{min}) = 0 \tag{5.3}$$

$$H(x_{max}) = 90 \tag{5.4}$$

Vrednosti H za ostale, vmesne senzorje se izračunajo po spodnji funkciji.

$$H(x) = \frac{(x - x_{min}) \cdot (H_{max} - H_{min})}{(x_{max} - x_{min})} + H_{min} \tag{5.5}$$

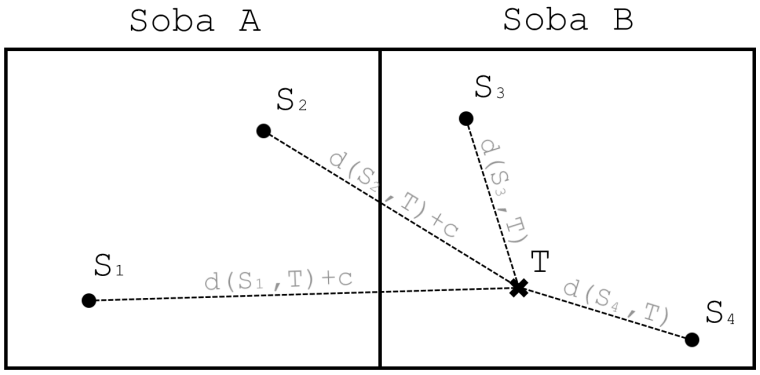
Če je uporabnik ročno vnesel najmanjšo in največjo vrednost legende, se uporabi enaka funkcija, le da se za spremenljivki x_{min} in x_{max} namesto minimuma in maksimuma uporabijo vrednosti, ki jih je vnesel uporabnik.

Za določanje barve točk med senzorji smo uporabili algoritem *IDW* (angl. Inverse Distance Weighting). Primeren je bil, ker se uporablja za računanje povprečne vrednosti 2D površine, kjer je znano število točk majhno [8]. V

našem primeru smo računali nekaj sto tisoč točk, kjer je znanih največ 6. Algoritem se lahko zapiše kot funkcija (5.6).

$$H_{IDW}(v) = \frac{\sum_{x \in V} \frac{H(x)}{d(v,x)^2}}{\sum_{x \in V} \frac{1}{d(v,x)^2}} \tag{5.6}$$

Ker je naša implementacija zahtevala še ločevanje sob, smo v zgoraj navedeni izraz dodali konstanto *insulation*, ki poveča razdaljo točke do senzorjev, ki niso v isti sobi. S povečevanjem in zmanjševanjem konstante *insulation*, smo dosegli učinek prehajanja toplote skozi stene, kot je prikazano na sliki 5.9. Konstanto *insulation* lahko uporabnik prilagodi v nastavitvah. Delovanje prilagojenega algoritma *IDW* je predstavljeno na sliki 5.8.

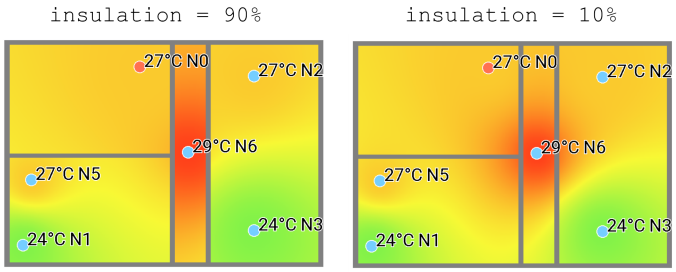


Slika 5.8: Delovanje algoritma na primeru dveh sob.

Rezultat funkcije H_{IDW} za točko T, bi bil za primer iz slike 5.8:

$$H_{IDW}(T) = \frac{\frac{H(S_1)}{(d(S_1,T)+c)^2} + \frac{H(S_2)}{(d(S_2,T)+c)^2} + \frac{H(S_3)}{(d(S_3,T))^2} + \frac{H(S_4)}{(d(S_4,T))^2}}{\frac{1}{(d(S_1,T)+c)^2} + \frac{1}{(d(S_2,T)+c)^2} + \frac{1}{(d(S_3,T))^2} + \frac{1}{(d(S_4,T))^2}} \tag{5.7}$$

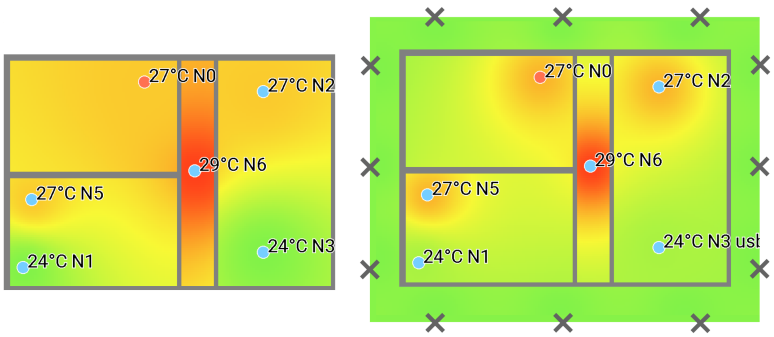
Konstanta *insulation* je uporabniku predstavljena v odstotkih, kjer 100% pomeni, da stene ne prepuščajo toplote oz. vlage. Z nekaj testiranja smo ugotovili, da je v rangu 0–200 sprememba konstante še najbolj opazna. Če



Slika 5.9: Procesirana slika za različni vrednosti *insulation*.

pa uporabnik drsnik pomakne do 100%, se konstanta nastavi na 2000, kjer prehajanje temperature skozi stene ni bilo več opazno.

Pri procesiranju smo dodali tudi možnost prikaza zunanje temperature. Podatki o zunanji temperaturi in vlagi se pridobijo iz programskega vmesnika Yahoo Weather API. Za prikaz zunanje temperature smo v množico senzorjev dodali 12 navideznih senzorjev na robove slike kot je označeno s križci na sliki 5.10. Kot je tudi prikazano na sliki, dodatek navideznih senzorjev na zunanjo stran vpliva na izris notranjosti, saj je bila konstanta *insulation* pri tem izrisu manj od 100%. Prikaz zunanje temperature je možno vklopiti in izklopiti v nastavitvah.



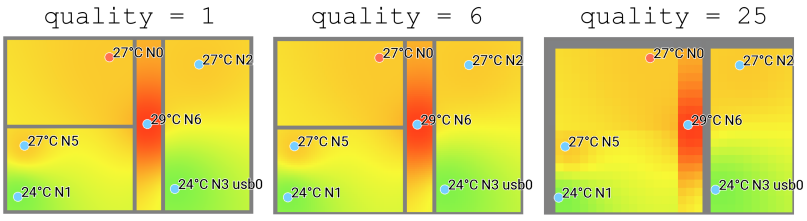
Slika 5.10: Procesirana slika brez in z zunanjo temperaturo.

Ker procesiranje slike tipično traja 10–20 sekund, smo vključili možnost

Velikost točke	Čas [ms]
1	5887
2	2161
3	1370
4	1081
5	1035
6	891
10	953
15	733
20	767

Tabela 5.1: Čas trajanja procesiranja za različne velikosti točke.

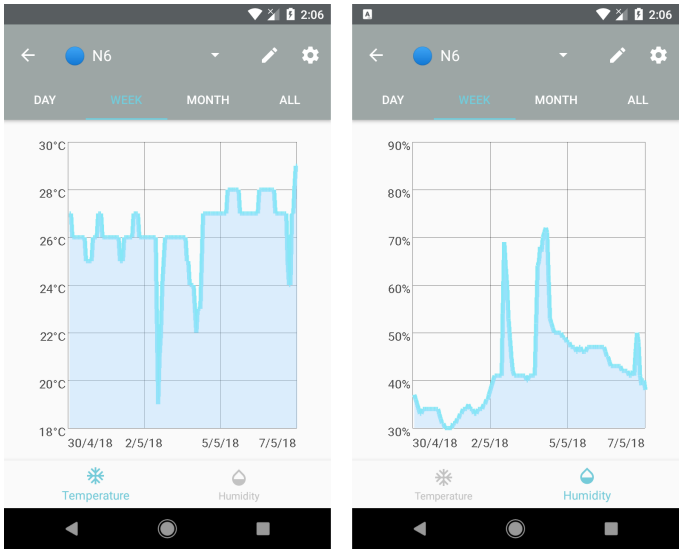
prilagajanja kvalitete procesirane slike. Prilagajanje kvalitete smo dodali samo v korak algoritma, ki zavzema največ (53%) celotnega procesiranja. Kvaliteta pri risanju vpliva na to, kako velika je posamezna točka, za katero se računa barva. Porabo časa za različne kvalitete smo merili s funkcijo „System.nanoTime()“ in čas pretvorili v milisekunde. Testirali smo privzeto sliko tlorisa, ki je velika 300×225 točk. V tabeli 5.1 so predstavljeni rezultati testiranja. Pri velikostih točk večjih od 4 je izguba kvalitete v našem primeru že bila opazna. Pri velikostih večjih od 10 pa so se že izgubile informacije o stenah, kot je prikazano na sliki 5.11. Na podlagi rezultatov smo omejili uporabnikovo izbiro na 1 do 4, saj bistvene izboljšave časa od 4 naprej nismo zabeležili.



Slika 5.11: Vpliv nastavitve kvalitete na izris procesirane slike.

Za risanje pik in besedila po sliki smo uporabili razred Canvas in njegove metode. Pika, ki se izriše na lokaciji senzorja je modre barve, če je senzor dosegljiv in rdeče, če ni. Besedilo smo risali s črno barvo in belim ozadjem, kar zagotavlja dobro vidljivost. Besedilo vsebuje enoto in ime senzorja, ki ga uporabnik lahko spremeni. Velikost izrisanih elementov se prilagaja glede na velikost slike, ki jo nastavi uporabnik.

5.4 Pregled zgodovine



Slika 5.12: Prikaz zgodovine z grafom.

Aktivnost „pregled zgodovine“ je namenjena prikazu zgodovine z grafom. Sestavljena je iz 4 komponent:

- statusna vrstica
- zavihki za obseg
- graf
- zavihka za vrsto prikaza

Zavihek v vmesniku	Obseg	API
DAY	Zadnjih 24 ur	gethourlyavg.php
WEEK	Zadnjih 7 dni	gethourlyavg.php
MONTH	Zadnjih 30 dni	getdailyavg.php
ALL	Od prve meritve naprej	getdailyavg.php

Tabela 5.2: Uporabljeni klici API za zavihke.

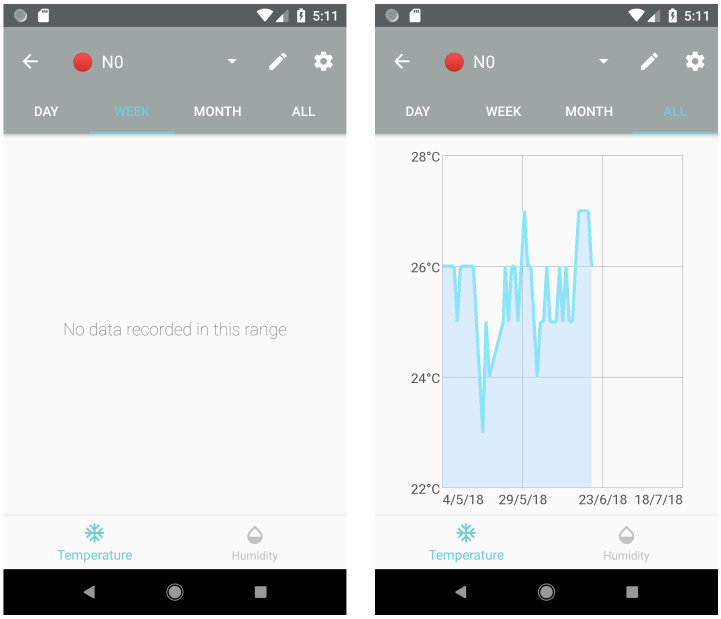
Statusna vrstica je enaka kot v aktivnosti „konfiguracija pozicij senzorjev“. Vsebuje spustni seznam za izbiro senzorja in gumb za preimenovanje senzorja. Osrednji element aktivnosti je graf. Za risanje grafa smo uporabili odprtokodno knjižnico „GraphView“ [21]. Na dnu sta še zavihka, ki omogočata preklapljanje med grafom temperature in grafom vlage. Izbira obsega grafa je možna preko štirih zavihkov na vrhu ali pa s potegom prsta levo ali desno. To gesto omogoča gradnik „viewpager“ [19]. Ob začetku aktivnosti se podatki prenesejo iz strežnika, nato pa se zgradijo grafi. Ob menjavi zavihkov se naslednji zavihek vedno zgradi v ozadju, da uporabniku ni potrebno čakati na izris grafa. Zadnji ogledani senzor, obseg grafa in izbrano veličino shranimo v nastavitve. Tako se tudi ob naslednjem zagonu aplikacije odpre enak graf. Vsak od zavihkov uporablja API klic, ki najbolj ustreza njegovemu obsegu. V tabeli 5.2 je našteto kateri zavihek uporablja kateri klic.

5.4.1 Knjižnica GraphView

Za implementacijo knjižnice „GraphView“ je bilo potrebno dodati spodnjo vrstico v datoteko build.gradle.

```
dependencies {
    ...
    implementation 'com.jjoe64:graphview:4.0.1'
}
```

Uporabili smo starejšo verzijo 4.0.1, ker je novejša verzija 4.2.2 imela težave



Slika 5.13: Prikaz grafa z manjkajočimi podatki.

pri izrisu napisov. Na vsako os smo želeli izpisati podatke po meri. Na osi X pri vlagi odstotek, pri temperaturi pa stopinje celzija. Tudi za prikaz datuma oziroma ure na X osi smo naredili slovenski format po meri. Knjižnica ponuja več različnih vrst grafov, najbolj primeren za našo uporabo je bil „LineGraph“.

Knjižnica deluje tako, da je prejete podatke najprej potrebno shraniti v vrsto, nato pa to vrsto podamo kot parameter pri risanju grafa. Klic API podatke vrne v formatu JSON in sicer vedno vrne vse podatke. Pri prebiranju podatkov iz JSON v vrsto, smo podatke tudi filtrirali glede na izbrani zavihek (dan, teden, ...). Vrsta vsebuje elemente podatkovnega tipa „DataPoint“, ki vsebujejo čas in podatek.
Deklaracija vrste:

```
LineGraphSeries<DataPoint> series = new LineGraphSeries<>();
```

Podatkovni tip „DataPoint“ pričakuje datum kot java razred „date“, zato je

bilo potrebno niz datuma iz spletnega programskega vmesnika API najprej pretvoriti, kot je navedeno spodaj.

```
format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
date = format.parse(timestamp);
series.appendData(new DataPoint(date, value)
```

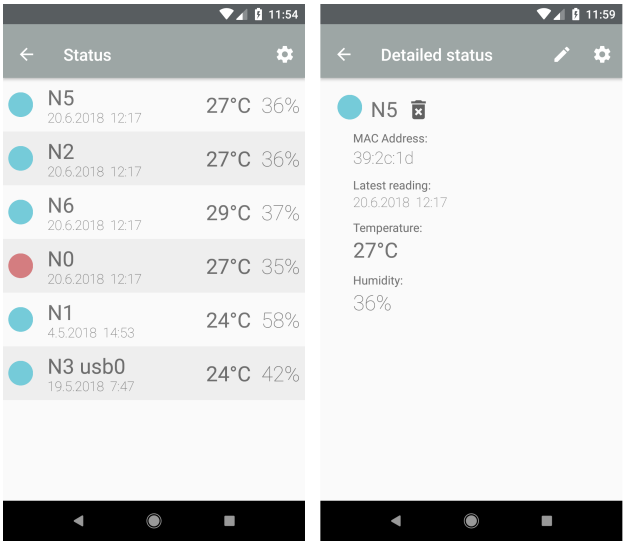
Če podatkov v zahtevanem obsegu ni, aplikacija grafa ne izriše, temveč prikaže sporočilo, da podatkov ni. Če od neke točke naprej podatkov ni, se izpiše prazen prostor kot je prikazano na sliki 5.13. Če se podatki spet pojavijo, bo izrisana ravna črta med zadnjim in novim podatkom.

5.5 Statusni pogled

Aktivnost „statusni pogled“ je namenjena hitremu pregledu senzorjev. Dodali smo jo, ker je procesiranje slike pri aktivnosti „pregled sob“ počasno. Uporabnik včasih želi hiter pregled nad vsemi senzorji. Aktivnost je sestavljena iz dveh delov. V prvem se s pomočjo gradnika „RecyclerView“ zgradi seznam iz katerega lahko uporabnik izbere senzor. Drugi del je nova aktivnost, ki se odpre s klikom na posamezne elemente seznama. V drugi aktivnosti vidimo nekaj več podatkov za posamezne elemente seznama prve aktivnosti.

Gradnik „RecyclerView“ je namenjen učinkovitemu prikazu elementov v seznamu. Med drugim omogoča polno prilagajanje vrstic seznama [17], zato smo ga uporabili. V vsaki vrstici smo navedli sledeče podatke:

- uporabniško ime senzorja
- datum in ura zadnjega branja
- temperatura
- vlaga



Slika 5.14: Različna stanja statusnega pogleda.

S klikom na element seznama se odpre druga aktivnost, ki vsebuje vse podatke iz seznama in še spodaj navedene:

- MAC naslov senzorja
- preimenovanje senzorja
- brisanje senzorja in njegovih meritev

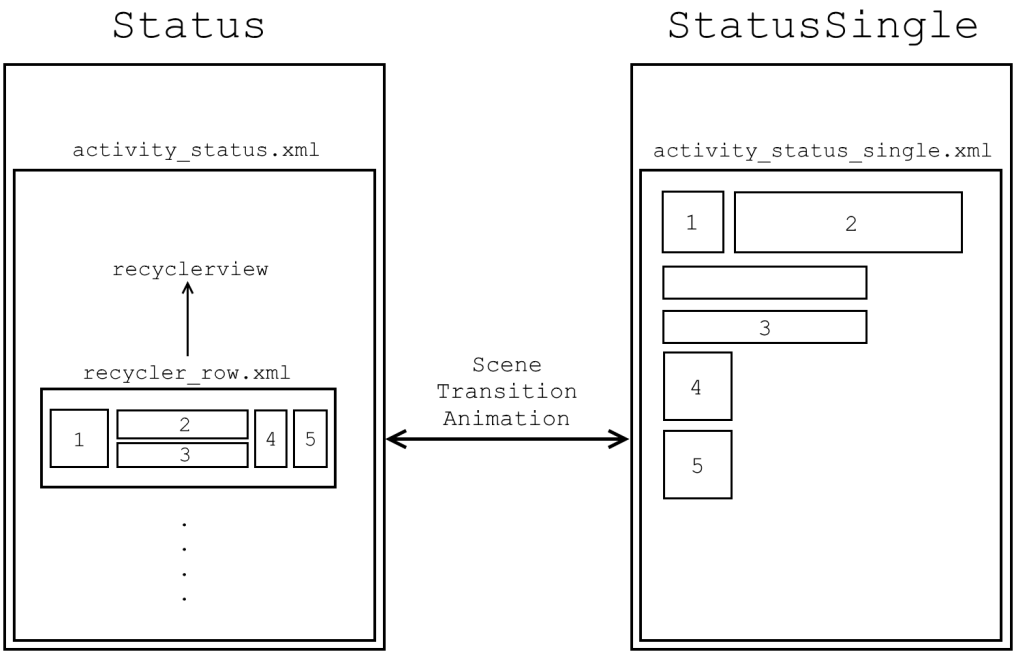
Ob kliku na ikono za brisanje se odpre pojavno okno, kjer lahko uporabnik brisanje potrdi ali zavrne. Obe aktivnosti sta prikazani na sliki 5.14.

5.5.1 Prehodna animacija

Prehod med dvema aktivnostma, ki sestavljata „statusni pogled“ smo animirali. Za animacijo smo uporabili „SceneTransitionAnimation“. Najprej smo znotraj datotek „activity_status.xml“ in „activity_status_single.xml“ gradnikom, ki smo jih animirali dodali parametre „android:transitionName“. Animirani gradniki so na sliki 5.15 označeni s številkami. V aplikaciji smo ustva-

rili par za vseh 5 gradnikov. Spodaj je primer za enega od animiranih gradnikov.

```
ActivityOptionsCompat options = ActivityOptionsCompat.
makeSceneTransitionAnimation(this, Pair.create(title,
ViewCompat.getTransitionName(title)));
startActivity(intent, options.toBundle());
```



Slika 5.15: Delovanje prehodne animacije v aktivnosti „statusni pogled“.

Animacija deluje tako, da se vsak gradnik iz prejšnjega mesta v ravni črti pomakne na novo mesto. Ko se uporabnik vrne nazaj, se animacija predvaja v obratni smeri. Na primeru slike 5.15 se 1 na levem zaslonu premakne na pozicijo 1 na desnem.

5.6 Nastavitve

Za shranjevanje nastavitev v aplikaciji smo uporabili vmesnik „SharedPreferences“. Vmesnik nastavitve shranjuje v notranji pomnilnik in zagotavlja, da vrednosti ostajajo konsistentne [23]. Vmesnik ustvari datoteko XML v notranjem pomnilniku aplikacije. V našem primeru je to bila SETTINGS.XML, ki se nahaja v mapi \data\data\com.example.nejc.smarthome\shared_prefs v notranjem pomnilniku naprave.

Primer branja z uporabo vmesnika SharedPreferences:

```
SharedPreferences pref = getSharedPreferences("SETTINGS", MODE_PRIVATE);
String url = pref.getString("webServer", "https://127.0.0.1");
```

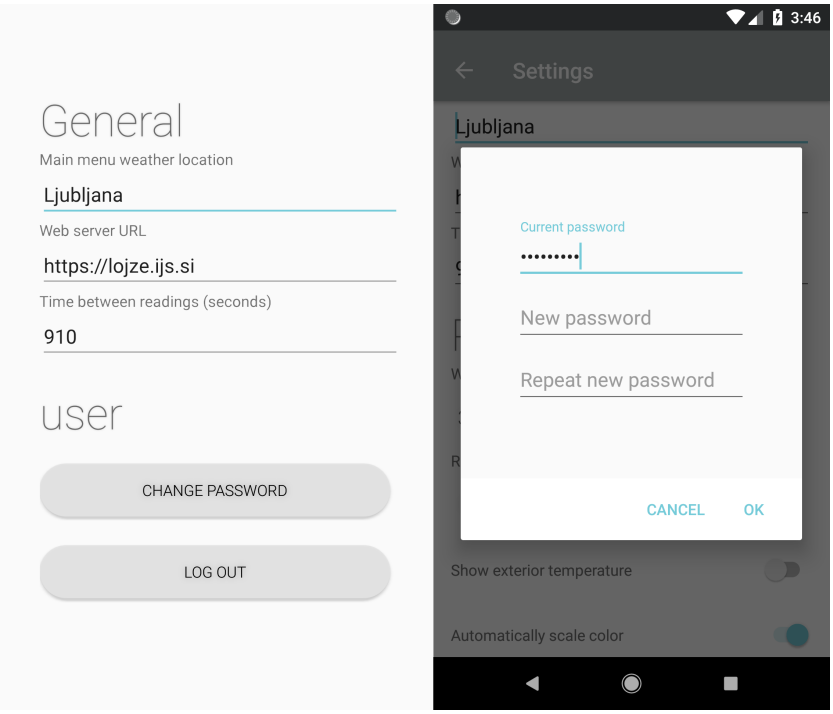
Primer pisanja z uporabo vmesnika SharedPreferences:

```
SharedPreferences.Editor editor = getSharedPreferences("SETTINGS",
MODE_PRIVATE).edit();
editor.putString("webServer", userinput.getText().toString() );
editor.apply();
```

Za aktivnost „nastavitve“ smo uporabili drsni seznam, saj je nastavitev več, kot bi jih lahko spravili na en zaslon. Drsni seznam je uporaben tudi zato, ker je uporaben v vertikalni in horizontalni postavitvi naprav. Nastavitve smo razdelili na 3 kategorije. Vsaka kategorija na drsnem seznamu je označena z velikim naslovom ob začetku kategorije.

5.6.1 Kategorija „splošno“

Kategorija „splošno“ vsebuje tri tekstovna polja. Prvo polje je ime mesta, ki se išče v programskem vmesniku Yahoo Weather API za prikaz vremena skozi celotno aplikacijo. Drugo polje je naslov strežnika Raspberry Pi, kjer je na voljo spletni programski vmesnik API, ki smo ga razvili. Tretje polje je čas osveževanja. Ta čas pove skripti Python na strežniku koliko časa naj mine med branji iz brezžičnih senzorjev.



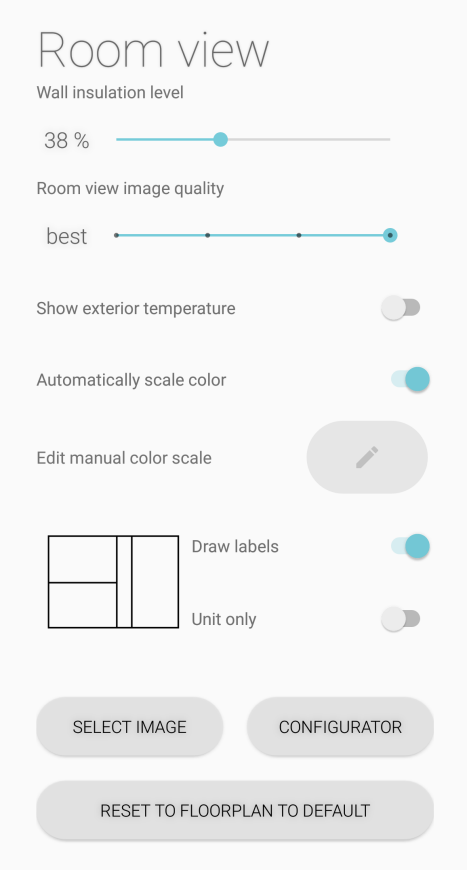
Slika 5.16: Kategoriji „splošno“ in „uporabnik“ v nastavitvah.

5.6.2 Kategorija „uporabnik“

Naslov kategorije se spreminja glede na prijavljenega uporabnika. V sliki 5.16 je prijavljen uporabnik „user“. V kategoriji sta samo dva gumba. Prvi gumb odpre pojavno okno za spremembo gesla, kot je prikazano na sliki 5.16. Drugi gumb uporabnika odjavi in ponovno odpre začetnega vodiča za konfiguracijo.

5.6.3 Kategorija „pregled sob“

Ta kategorija je namenjeni nastavitvam, ki se nanašajo na aktivnost „pregled sob“. Kategorija je prikazana na sliki 5.18. Prvi dve nastavitvi sta drsnika. Za predstavitev izolacije smo uporabili odstotke, za kar je primeren navadni drsnik, ki ima ravno 100 vrednosti. Za predstavitev kvalitete smo uporabili diskretni drsnik s 4 nivoji. Kvalitete so poimenovane opisno z angleško besedo (worst, low, high, best).

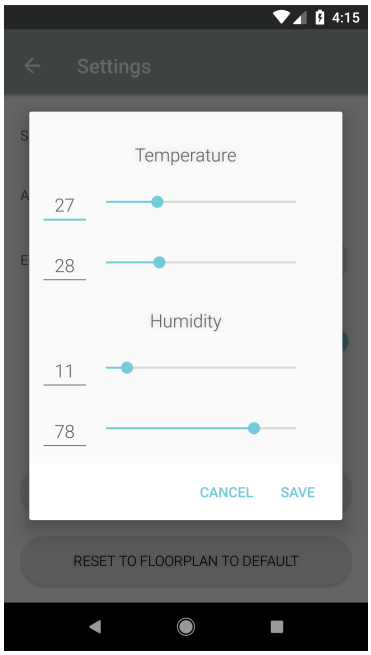


Slika 5.17: Kategorija „pregled sob“ v nastavitvah.

Spodaj so našteti gumbi znotraj kategorije „pregled sob“:

- „Show exterior temperature“ obarva zunanji del stavbe s temperaturo pridobljeno iz programskega vmesnika Yahoo Weather API
- „Automatically scale color“ določa ali so robovi legende določeni avtomatsko ali ročno
- „Draw labels“ prikaže in skrije napise na procesirani sliki.
- „Unit only“ na procesirani sliki skrije imena senzorjev in prikaže samo vrednosti

- „Select image“ odpre zunanjo aplikacijo za izbiro slike tlorisa
- „Configurator“ odpre aktivnost „konfiguracija pozicij senzorjev“
- „Reset floorplan to default“ izbriše uporabniško sliko tlorisa in uporabi privzeto, ki je vgrajena v aplikacijo



Slika 5.18: Ročna konfiguracija robov legende.

Ročna konfiguracija robov legende je prikazana na sliki 5.18. Pri ročni konfiguraciji smo pazili, da uporabnik ne more pomotoma vnesti nelogičnih vrednosti. Minimum tako ne more biti večji kot maksimum, saj se v tem primeru maksimum avtomatsko poveča. Upravljanje z drsniki je hitro, ima pa pomanjkljivost, da je pri uporabniških vmesnikih na dotik nenatančno. Zato smo dodali na levo stran drsnika tekstovno polje, ki prikazuje vrednost drsnika. Vanj pa je mogoče tudi direktno vpisati številke. Pojavno okno za ročno konfiguracijo legende se odpre z gumbom „Edit manual color scale“. Kadar je konfiguracija nastavljena na avtomatsko, gumb osivi.

Poglavje 6

Sklepne ugotovitve

V diplomski nalogi smo se spoznali z delovanjem multi-hop mehanizma za BLE razvitega na IJS. S to temo sem se podrobno spoznal med opravljanjem obvezne delovne prakse na institutu „Jožef Stefan“. V okviru diplome smo nadgradili brezžične senzorje z možnostjo merjenja vlage. Razvili smo programsko opremo za Raspberry Pi, ki podatke prejema iz brezžičnega senzorskega omrežja in jih hrani v podatkovni zbirki. Za strežnik smo razvili tudi spletni programski vmesnik API.

Razvili smo delujočo mobilno aplikacijo za senzorsko omrežje, ki ga je bilo prej možno spremljati le lokalno preko vmesnika UART. Mobilna aplikacija omogoča pregled nad stanjem senzorjev preko interneta. To omogoča uporabnikom vpogled v stanje stavbe na daljavo. Prikaz zgodovine v aplikaciji omogoča uporabnikom analizo nad posameznimi sobami. Na podlagi grafov se uporabnik lahko odloči, če katero sobo pretirano hladi ali ogreva. Prikaz temperature oziroma vlage z barvami na tlorisu omogoča hiter pregled nad trenutnim stanjem v stavbi. Če ena izmed sob močno odstopa od drugih se to z barvo hitro opazi. Uporaben je tudi sistem, ki smo ga razvili za prilagajanje tlorisa. Senzorje je mogoče postaviti v katero koli stavbo, nato pa se tloris in postavitev senzorjev lahko prilagodi s pomočjo aplikacije. V aplikacijo smo vključili tudi pogled zgodovinskih podatkov z grafom, ki omogoča prilagajanje obsega grafa.

Pri pregledu sob smo algoritem za procesiranje slike sicer izboljšali, vendar je še vedno počasen. Največ časa sedaj porabi algoritem *flood fill*, ki ustrezno zazna posamezne sobe samo na podlagi rastrske slike. Možna izboljšava bi bila, da ne uporabljamo več rastrske slike, temveč v aplikacijo vključimo konfigurator tlorisa, ki posamezne sobe definira na podlagi koordinat. Tako bi se izognili potrebi po zaznavanju sob in edini čas za procesiranje bi porabilo dejansko risanje barv. Z izboljšavami risanje barv traja tudi manj kot sekundo. Druga rešitev za isti problem bi bila, da se slike periodično procesira na strežniku in se ob zahtevi uporabniku samo pošlje slika. Tako uporabniku ne bi bilo treba čakati na procesiranje slike.

Ko je bil sistem postavljen, smo pustili da teče in zbira podatke. Po nekaj tednih se je pojavila težava s skripto za zajemanje podatkov na strežnik. Ustavila se je vsakič, ko je mikroračunalnik Raspberry Pi izgubil napajanje. To smo rešili tako, da smo zagotovili da se skripta zažene ob zagonu operacijskega sistema. Od takrat naprej skripta deluje ustrezno in na mobilni aplikaciji so podatki vedno na voljo.

Literatura

[1] Bluegiga. BLE113 DATA SHEET. Version 1.2, nov. 2013.

[2] Bluegiga. BGSCRIPT SCRIPTING LANGUAGE. Version 3.6, maj. 2014.

[3] Skočir Branko. Multi-hop communication in Bluetooth Low Energy ad hoc network . Magistrska naloga, Mednarodna podiplomska šola Jožefa Stefana, Ljubljana, 2014.

[4] NXP B.V. LM75B Digital temperature sensor and thermal watchdog. Rev. 6.1, feb. 2015.

[5] Carles Gomez, Joaquim Oller, and Josep Paradells. Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. *Sensors*, 12(9):11734–11753, 2012.

[6] Honeywell. I^2C Communication with the Honeywell HumidIcon^{TM} Digital Humidity/Temperature Sensors. 009061-2-EN, jun. 2012.

[7] Honeywell. HumidIcon^{TM} Digital Humidity/Temperature Sensors. 009059-7-EN, maj. 2015.

[8] Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM national conference*, pages 517–524. ACM, 1968.

[9] Branko Skočir. Multi-hop communication in bluetooth low energy adhoc wireless sensor network. *Informacije MIDEM*, 48(2):85–96, 2018.

[10] Branko Skočir, Gregor Papa, and Anton Biasizzo. Multi-hop communication code. IJS technical report No. 12380, 2018.

[11] Slika modula BLE113. Dosegljivo: https://siliconlabs-h.assetsadobe.com/is/image//content/dam/siliconlabs/images/products/Bluetooth/bluetooth_low_energy/bluegiga_bluetooth_low_energy_modules/ble113-module-2.jpg. [Dostopano: 15. 6. 2018].

[12] Slika računalnika Raspberry Pi 2B. Dosegljivo: https://upload.wikimedia.org/wikipedia/commons/thumb/d/d4/Raspberry-Pi-2-Bare-BR.jpg/1920px-Raspberry-Pi-2-Bare-BR.jpg. [Dostopano: 15. 6. 2018].

[13] Slika senzorja HIH6130-021-001. Dosegljivo: https://www.onlinecomponents.com/images/Parts/LargeImages/43883000.jpg. [Dostopano: 1. 7. 2018].

[14] Slika senzorja LM75B. Dosegljivo: https://i.ebayimg.com/images/g/Zm4AAMXQwKdRb0Yy/s-l300.jpg. [Dostopano: 1. 7. 2018].

[15] RASPBERRY PI 2 MODEL B. Dosegljivo: https://www.raspberrypi.org/products/raspberry-pi-2-model-b/, 2015. [Dostopano: 1. 7. 2018].

[16] Android developers: AsyncTask. Dosegljivo: https://developer.android.com/reference/android/os/AsyncTask, 2018. [Dostopano: 8. 7. 2018].

[17] Android developers: RecyclerView. Dosegljivo: https://developer.android.com/guide/topics/ui/layout/recyclerview, 2018. [Dostopano: 16. 7. 2018].

[18] Android developers: Toasts overview. Dosegljivo: https://developer.android.com/guide/topics/ui/notifiers/toasts, 2018. [Dostopano: 10. 7. 2018].

[19] Android developers: ViewPager. Dosegljivo: https://developer.android.com/reference/android/support/v4/view/ViewPager, 2018. [Dostopano: 11. 7. 2018].

[20] Android developers: Volley. Dosegljivo: https://developer.android.com/training/volley/, 2018. [Dostopano: 8. 7. 2018].

[21] GraphView - open source graph plotting library for Android. Dosegljivo: http://www.android-graphview.org/, 2018. [Dostopano: 15. 7. 2018].

[22] RASPBIAN. Dosegljivo: https://www.raspberrypi.org/downloads/raspbian/, 2018. [Dostopano: 3. 7. 2018].

[23] SharedPreferences. Dosegljivo: https://developer.android.com/reference/android/content/SharedPreferences, 2018. [Dostopano: 8. 7. 2018].